Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

IE0499 – Proyecto Eléctrico

HDMX early bird

por

Felipe Badilla Marchena

Ciudad Universitaria Rodrigo Facio

Abril de 2022

HDMX early bird

por

Felipe Badilla Marchena

B70848

IE0499 – Proyecto Eléctrico

Aprobado por

Ing. Marco Villalta
Profesor guía

Ing. Enrique Coen
Profesor lector

Ing. Dagoberto Vasquez
Profesor lector

Abril de 2022

Resumen

HDMX early bird

por

Felipe Badilla Marchena

Universidad de Costa Rica
Escuela de Ingeniería Eléctrica
Profesor guía: Ing. Marco Villalta
Abril de 2022

Este trabajo explora las posibilidades que tiene el algoritmo de aprendizaje automático Naive Bayes y sus variantes para predecir los daños causados por el desgaste de la partes de la máquina HDMX utilizada por intel en su proceso de pruebas de procesadores.

Se transforman los datos de tal manera que pasan de ser datos descriptivos de una prueba realizada a un procesador a ser datos descriptivos de los posibles puntos de falla de una máquina y así sean procesables por un algoritmo de aprendizaje automático y se evalúan tres variantes del algoritmo Naive Bayes basado en el teorema de Bayes el cual con su su simpleza de operación puede llegar a tener grandes resultados.

Luego de un proceso de selección basado en indicadores de rendimiento del algoritmo de aprendizaje automático se concluye que el más óptimo es el llamado multinomial que promete hasta un 80 % de rendimiento en sus clasificaciones de una parte de la máquina como buena o marginal.

Los resultados al poner a prueba este algoritmo son un tanto menores a ellos si se toma en cuenta solo las partes señaladas como malas obteniendo 3/5 predicciones correcta sin embargo para los alcances de este proyecto no es posible medir que tan certero es en sus predicciones de que una parte está en buen estado, se concluye que este algoritmo puede ser de momento una herramienta de apoyo buena en conjunto con otras herramientas internas y de esta forma se estaría usando como un indicador de una posible falla que hay que evaluar si debe ser aceptado o rechazado.

***Palabras claves**: Aprendizaje automático, Naive Bayes, Análisis de datos.*

Abstract

HDMX early bird

Original in Spanish. Translated as: “Title of the Project in English”

by

Felipe Badilla Marchena

University of Costa Rica
Department of Electrical Engineering
Tutor: Ing. Marco Villalta
April of 2022

This work explores the possibilities of the Naive Bayes machine learning algorithm and its variants to predict damage caused by wear of HDMX machine parts used by intel in its processor testing process. The data is transformed in such a way that it goes from being descriptive data of a test carried out to a processor to be descriptive data of the possible failure points of a machine and thus be actionable by a machine learning algorithm and three variants of the Naive algorithm are evaluated Bayes based on the Bayes theorem which with its simplicity of operation can end up in Big results.

After a selection process based on performance indicators of the MAchine learning algorithm we is concluded that the most optimal is the multinomial model that promises up to 80 % of performance in their ratings of a machine part as good or marginal.

The real results when testing this algorithm are somewhat lower than them if one takes into consideration only the parts marked as bad obtaining 3/5 correct predictions however for the scopes of this project it is not possible to measure how accurate it is in its predictions that a part is in good condition, it is concluded that this algorithm can be a good support tool and in conjunction with other internal tools and in this way it would be used as an indicator of a possible fault that must be evaluated if it should be accepted or rejected.

***Keywords**: Machine learning, Naive Bayes, Data analysis.*

Índice general

Índice de figuras

Capítulo 1

Introducción

El presente documento contiene el desarrollo del proyecto HDMX early bird traducido al español como HDMX pájaro mañanero, el cual pretende mejorar la eficiencia en uno de los procesos de producción de procesadores fabricados por la empresa Intel, esto mediante de la detección temprana de partes defectuosas dentro de las máquina que prueban los procesadores, evitando desechar unidades en buenas condiciones clasificadas de otra forma por mal funcionamiento de la maquinaria de pruebas.

Intel es una de las empresas líderes de mercado en cuanto a desarrollo y producción de procesadores, en la sede de Costa Rica se fabrican procesadores de alto rendimiento los cuales tienen que pasar por una serie de operaciones que aseguran la integridad, calidad y correcta funcionalidad de cada uno de los procesadores.

1.1. ¿Qué es HDMX?

Dentro de las operaciones de Intel en Costa Rica existe una diseñada para probar el rendimiento, integridad funcional y estructural de las unidades de procesamiento. Esta operación de la que hablamos es llevada a cabo mediante una máquina llamada high density module (la x hace referencia al número de máquina) o más conocida como HDMX. La HDMX es una máquina segmentada la cual es capaz de probar 30 unidades al mismo tiempo en los espacios llamados "celdas", este concepto de celda toma importancia dentro de este proyecto ya que ahí es donde queremos detectar un posible fallo.

Las celdas son los espacios de prueba de las unidades y esta cuenta con tres partes extraíbles que son los puntos de fallo más comunes ya que cuentan con mucho desgaste en el proceso de pruebas. Estas tres son: el socket (nos referiremos a él como C1) que es donde se coloca el procesador, el *heater* que presiona la unidad contra el socket y maneja la temperatura (lo llamaremos C2) y por último el C3 coordina a C1 y C2 en el proceso y secuencia de pruebas, es importante tomar en cuenta que estas partes son usualmente llamadas colaterales y a veces se refiere a ellos de esta manera en el documento.

1.2. Alcances

Se pretende escribir un algoritmo de inteligencia artificial que sea capaz de clasificar cada una de estas tres partes dentro de una celda como marginal o funcional de acuerdo a su comportamiento reflejado en las entradas que serán extraídas de los datos tomados de las pruebas y subidos a las bases de datos de Intel.

En este sentido se supondrá que ya se cuenta con un archivo de excel con los datos en una carpeta específica donde se actualizará cada cierto tiempo y por tanto no se toma en cuenta la parte de la extracción de datos de las bases.

Intel es una compañía muy grande y con muchos productos a la venta los cuales requieren de equipos diferentes para realizar pruebas, por lo que para efectos de este proyecto se trabajará solo con uno de los productos en cual actualmente se trabaja con 2-3 maquinas HDMX pero se simulará como máximo con una máquina.

Esta máquina tiene diferentes códigos de fallo como salida que indican cuál es el fallo específico de cada procesador sin embargo se ignorará para ciertas partes del análisis el tipo de fallo y se tomará como un dato binario, bueno o malo.

En la última parte de este proyecto se realizará una visualización simple de los resultados con las tres partes extraíbles y su ubicación en la máquina (celda).

1.3. Justificación

Como se menciona anteriormente, Intel es una de las empresas mejor posicionadas en el comercio de procesadores para muchas aplicaciones. Como tal para lograr ser una empresa rentable y competitiva en el mercado es necesario que se reduzcan los costos de producción a lo menor posible. En este contexto debido a las fallas de hardware de pruebas hacia los procesadores se pueden perder muchas unidades por falsos negativos en donde se marcan como malas pero es a causa de mal funcionamiento de la máquina que las prueba, la intensión de este proyecto es minimizar ese impacto mediante la detección temprana de estas fallas.

En este sentido con la detección temprana se lograrán más objetivos como el de minimizar el tiempo de análisis de datos que hace un ingeniero rutinariamente, por que este podrá dedicarle más tiempo a distintos proyectos de mejora, además que se pueden lograr mejores resultados con menos recursos de operación.

1.4. Descripción del problema

Dentro el proceso de prueba de los procesadores la máquina tiene una salida con los resultados de las pruebas en forma de códigos donde cada código nos indica el fallo o éxito de la pruebas y también la posible causa del fallo, a estos códigos les llamamos bines y es de suma importancia que estas pruebas sean veraces y no falsos positivos. Los falsos positivos llamamos a un bin de fallo pero provocado por la máquina, esto genera un impacto económico y un gasto de tiempo adicional porque se debe probar

varias veces la misma unidad, aparte que esto es uno de los indicadores que se toman en la fábrica para medir el éxito de la operación.

Con el objetivo de minimizar el impacto provocado por mal funcionamiento en la máquina se pretende implementar un algoritmo que analice y detecte posibles fallos en la máquina en el menor tiempo posible y con la menor intervención humana necesaria durante la detección de una parte mala y en cambio solo influir en la toma de decisión y acción de reparación de la parte para hacer los riesgos más pequeños.

Se escribirá un algoritmo de aprendizaje automático o *machine learnig* de tipo clasificatorio cuyo nombre es "Naive Bayes", esto en el lenguaje de programación python y con la ayuda de la librería Scikit-learn para el preprocesamiento de datos y la construcción del modelo de inteligencia artificial. Por último se construirá una salida visual sencilla y de fácil entendimiento para el usuario con códigos de colores para su rápido entendimiento.

1.5. Objetivos

1.5.1. Objetivos Generales

- Detectar partes defectuosas dentro la máquina HDMX para tomar acción rápida y así minimizar el impacto económico y temporal en el proceso de prueba de procesadores en producción.
- Facilitar y eficientizar el proceso de análisis de datos por parte de los ingenieros a cargo de la máquina y aparte minimizar los factores de error en el proceso de análisis manual.

1.5.2. Objetivos específicos

- Realizar un preprocesamiento de los datos provenientes de una hoja de excel para extraer las características específicas necesarias para la entrada del modelo que realiza la evaluación probabilística.
- Aplicar el algoritmo de machine learning "Naive Bayes" para asociar y calcular las probabilidades de todos los parámetros que se evalúan dentro de una misma celda y un colateral en particular.
- Crear una salida de resultados al usuario que muestre los datos necesarios y que permita una comprensión fácil y rápida.
- Evaluar los resultados con la toma de acciones experimentales en la máquina física según los resultados del algoritmo programado para comprobar la efectividad del mismo.

1.6. Metodología

Para ejecutar este proyecto se realiza una previa investigación acerca de los diferentes tipos de algoritmos de *machine learning* y sus usos en donde se toma la decisión de implementar el naive bayes debido a la naturaleza de los datos y necesidad de salida de los mismos.

Una de las partes más cruciales de este proyecto es el preprocesamiento de los datos ya que estos serán la entrada del algoritmo de *machine learning* que a su vez darán pie al modelo que será formado para predecir el comportamiento marginal de los colaterales.

Para la etapa de preprocesamiento de datos se acomodan y transformaran convenientemente para formar las entradas que se explican más adelante, esto mediante código en python y las librerías de pandas, scikit-learn y numpy.

Una vez se tiene listo y afinados los datos de entrada se procede a la construcción del modelo mediante *machine learning* con la librería de scikit-learn en donde se procederá a usar los tres tipos de naive bayes disponibles: Gaussian, Multinomial y Bernuilli, esto con el objetivo de evaluar el más adecuado para la naturaleza de los datos. Una vez se tengan resultados estables para los tres colaterales dentro de una celda se escalará a varias celdas dentro de una misma máquina.

Una vez se tenga todos los resultados de las celdas se tomará la información para enseñarla gráficamente mediante imágenes formadas con las librerías seaborn y matplotlib que nos facilitarán la interfaz de usuario para un mejor entendimiento y rápida toma de decisiones de parte de los ingenieros.

Por último y de las partes más importantes es la evaluación de los resultados que no solo se dará un análisis de eficiencia calculado mediante código sino que se evaluará la veracidad de las predicciones con datos a tiempo real donde se tomarán acciones en los colaterales de Intel para evaluar su eficiencia y posibles fallos, en caso de encontrar alguna falla en los mismos se tomará como verdadero para el cálculo de eficiencia experimental.

1.7. Calendarización

	Entregables	Acciones/ Investigación/ Pruebas
W05	Objetivos y Descripción	- Completar anteproyecto. - Definir el tipo de algoritmo de naive bayes. - Definir entradas.
W06	Anteproyecto Capitulo 1.	- Preprocesamiento de datos de entrada.
W07		- Modelar los datos con el algoritmo de ML. - Documentación avance 1.
W08	Avance 1 - Capítulo 2.	- Escalar a más celdas. - Troubleshooting.
W09	Presentación de un minuto.	- Escalar a más celdas. - Troubleshooting.
W10		- Presentación gráfica de los datos.
W11		- Piloto con datos a tiempo real. - Documentación.
W12		- Piloto con datos a tiempo real. - Documentación.
W13		- Comprobación de veracidad con arreglos en colaterales - Documentación.
W14		-Análisis de resultados. - Documentación.
W15	Borrador final completo y presentación preliminar.	- Ajuste de últimos detalles.
W16	Carta solicitud de presentación.	
W17	Presentación en línea.	
W18	Informe Final.	

Capítulo 2

Marco Teórico

En el área de pruebas de Intel se han creado distintas herramientas para facilitar al ingeniero el análisis y control de las máquinas de prueba a los procesadores, sin embargo estas se basan en su mayoría en algoritmos programados de experiencias previas. Dicho esto, este proyecto busca combinar estas experiencias previas junto con el *machine learning* alimentado por una cantidad grande de datos con el objetivo de buscar patrones diferentes en los indicios de marginalidades dentro de la máquina, para esto es importante conocer sobre los siguientes conceptos.

2.1. Aprendizaje Automático

Antes de definir qué entendemos por aprendizaje automático o *machine learning* (ML por sus siglas en inglés) es importante refrescar lo que hacemos en algoritmo de programación convencional. Cuando hablamos de la programación tradicional esto se refiere a un conjunto de pasos, instrucciones o reglas bien definidas y detalladas que describen como es que se resuelve un problema en particular, esto escrito en un lenguaje que después de un proceso de compilación pueda ser interpretado por un computador.

Este tipo de programación es muy útil para una gran cantidad de problemas sin embargo existe otra gran cantidad de escenarios en donde crear una guía paso a paso para resolver un problema puede volverse muy complicado debido al nivel de abstracción, subjetividad que conlleve el tema o la gran cantidad de variables que hacen del problema sumamente grande y con variaciones múltiples, unos ejemplos de esto sería hacer que una computadora entienda un documento escrito a mano ya que existen muchas variantes en las escritura entre personas y múltiples caracteres que detectar en una misma imagen. [4]

Continuando con el mismo ejemplo anterior sobre la detección del texto escrito a mano dentro de una imagen, supongamos que contamos con una gran cantidad de imágenes con muchos ejemplos de variaciones, todas debidamente marcadas con la letra o número que estas contienen por lo que ya tenemos un extenso catálogo de ejemplos etiquetados que pretenden mostrar como debe comportarse el programa que estamos diseñando. Tratar de llevar a cabo esta tarea con el método tradicional puede ser llegar a complicarse mucho tomando en cuenta la gran cantidad de variaciones, inclusive las que no se ven reflejadas en los ejemplos que tenemos dentro del catálogo. He aquí donde entran los algoritmos de *Machine Learning* ya que estos no requieren de un diseño específico y detallado, estas resuelven

problemas de una manera más genérica. Los algoritmos de ML son capaces de aprender de una gran cantidad de información de entrada debidamente etiquetada para lograr predecir correctamente en situaciones futuras, y con aprender nos referimos a que internamente desarrollan un modelo o reglas a partir de los datos de los que esta se alimenta. [4]

Existen múltiples técnicas de aprendizaje con la cual los algoritmos pueden generar un modelo para predecir un resultados, los más comunes son:

2.1.1. Aprendizaje supervisado

El aprendizaje supervisado o SL por sus siglas en ingles en un método en el cual los datos que le son suministrados ya se encuentran previamente etiquetados, esto quiere decir que cada uno tiene la "respuesta correcta" de como debería comportarse el algoritmo, siguiendo con el mismo ejemplo de archivo con texto escrito a mano podríamos suministrar una serie de imágenes de una misma letra escrita por diferentes manos pero las imágenes vienen etiquetadas con la letra que contienen. En este sentido el algoritmo debe reconocer las características claves para en un futuro predecir cuál letra será de manera correcta. [5] Algunos de los algoritmos de aprendizaje automático supervisado son:

clasificación

En el aprendizaje supervisado se habla de clasificación cuando las posibles etiquetas de clase toman valores de entre un conjunto discreto y definido para poder lograr una predicción. [6]

Regresión

En contrate con lo anterior, en caso de que los valores de las posible etiquetas sean continuos se trata de un problema de regresión el que dará pie al modelo de predicción. [6]

2.1.2. Aprendizaje no supervisado

A diferencia del aprendizaje supervisado, el no supervisado es alimentado con datos que no están etiquetados o en otras palabra no tienen esa respuesta de la salida esperada del algoritmo por lo tanto en nuestro ejemplo de la imágenes de texto escrito a mano, estos datos de alimentación solo serían imágenes sin decir a que letra corresponde. Con esta información el algoritmo debe ser capaz de identificar tendencias, grupos u otro similitud entre la información. [5] Algunos de los algoritmos de aprendizaje no supervisado son:

Segmentación (*clustering*)

La técnica de segmentación es una de las más populares y utilizadas de los algoritmos de aprendizaje no supervisado. Esta consiste en el agrupamiento de datos de acuerdo a su similitud sin un imposición previa de restricciones por parte del programador. Esto se logra mediante la inserción de datos descriptores que mostrarán patrones. [3]

Reducción de dimensionalidad

Esta técnica crea una representación de los datos originales en una dimensión más pequeña a la inicial, con la gran ventaja de seleccionar los patrones o características más significativas de los datos en alta dimensión. [1]

2.1.3. Aprendizaje profundo

El último tipo de aprendizaje del cual se da información es diferente a los dos anteriores, de hecho este es considerado de como un categoría aparte del *machine learning* tradicional. Es te método de aprendizaje de máquina hace referencia a las famosas redes neuronales. Una red neuronal de dos capas es un modelo que intenta asemejar un comportamiento real de una red neuronal pero sin embargo es un hecho que los grandes resultados de esta técnica vienen en cuanto se juntan múltiples capas de nodos o neuronas trabajando a la misma vez. [5] De este existen también subconjuntos de deep learning como perceptrón multicapa (MLP), red neuronales convolucionales (CNN) y redes neuronales recurrente (RNN).

2.2. Naive Bayes

Naive Bayes o bayes incente en español es un algoritmo de aprendizaje automático que tiene como base el teorema de Bayes, de ahí su nombre. Este algoritmo es ideal para decisiones binarias con datos bien etiquetados como se pretende usar en este proyecto. Una de las aplicaciones más conocidas en este algoritmo es la de clasificación de correos electrónicos como veraces o SPAM. De este teorema, aplicaciones y más se desarrollará a continuación.

2.2.1. Teorema de Bayes

El teorema de Bayes es uno de los temas más conocidos dentro de la teoría de la probabilidad, fue propuesto por Thomas Bayes quien tras una serie de análisis matemáticos expresa la relación probabilística que existe entre la probabilidad de un evento .ᴬ"dado que sucede Bƶ un evento "B"dado que sucede .ᴬ", dicha relación se expresa de la siguiente manera. [2]

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.1}$$

donde:

- $P(A)$ Es la probabilidad del evento A.
- $P(B)$ Es la probabilidad del evento B.
- $P(A|B)$ Es la probabilidad de A dado el evento B.
- $P(B|A)$ Es la probabilidad de B dado el evento A.

Esta formula puede ser expandida a múltiples condiciones para un evento particular lo cual lo hace muy atractivo para este y otros usos, ya que podemos tener un efecto A que es causado por múltiples causas o condiciones B y ahora queremos ver como estas condiciones influyen sobre el efecto final para calcular las probabilidades de que ocurra A y tomar decisiones. Algo importante de conversar con este teorema es la independencia de los datos en donde decimos que dos eventos A y B son independientes si se cumple que:

$$P(A|B) = P(A) \tag{2.2}$$

Y esto quiere decir que el evento B no afecta el evento A. Esta es la base de el clasificador ingenuo de Bayes, ahora bien, la razón por la que llamamos a este clasificador como ingenuo es que estamos tomando el supuesto de que todos nuestros datos son estadísticamente independiente aunque no siempre necesariamente lo sean. [8]

Este algoritmo suele ser muy atractivo debido a su simplicidad y escalabilidad, suele ser bastante efectivo en problemas binarios y multiclase. Aunque es bien sabido y mencionado por los autores que este clasificador da probabilidades muy poco precisas que no deben ser tomadas en cuenta, estas sí son de gran utilidad para poder tomar un criterio de decisión

2.2.2. Aplicaciones

Este algoritmo es usado en múltiples aplicaciones como por ejemplo en reconocimiento de escritura a mano, clasificación de productos, análisis bursátiles entre otros sin embargo el ejemplo más utilizado es el de un filtro de correos de spam dentro de nuestra bandeja de entrada en el correo electrónico, para este caso el algoritmo se encargará de clasificar el correo como spam o no en función de las palabras que este tenga en su contenido, para entrenar un algoritmo de este índole se le da una cantidad grande de datos clasificados (etiquetados) como spam que contendrán palabras similares y por tanto se podrán hacer predicciones bastante certeras.

2.3. Datos de entrada

Para un algoritmo de ML los datos lo son todo ya que el modelo generado que hará predicciones en base a sus entradas depende directamente de estos datos entonces a continuación se presentan algunos conceptos del campo de estudio que serán útiles más adelante en el desarrollo del proyecto

2.3.1. Fallo sólido

Como estamos hablando de un proceso de prueba de procesadores, las máquinas van a correr una serie de pruebas para verificar que la unidad funciona de manera correcta. Existe ocasiones en la máquina que lleva a cabo estas pruebas puede estar defectuosa e indique un falso negativo en la unidad, sin embargo este caso es bien previsto y por tanto si una unidad falla por cierta razón por lo general se le realiza una segunda prueba (puede ser más o incluso ninguna dependiendo del fallo), a estas oportunidades extras las llamamos *Retest* (retesteo traducido al español) y durante esta pueden darse varias situaciones, una de ellas sería que de bines de fallo diferentes a lo cual llamamos bin switching en inglés, también puede

ocurrir que la unidad pase en otro test o bien por último que siempre de el mismo fallo y por tanto esa unidad sí presenta ese fallo específico, a esta situación le llamamos fallo sólido.

2.3.2. Bines general/lote

Las unidades generalmente son clasificadas según lotes de producción que nos dan trazabilidad de fallos, un lote es un conjunto de unidades que fueron producidas bajo condiciones similares. Durante el preprocesamiento de datos estaremos recolectando datos de la cantidad de bines de fallo que presenta un colateral en todo su tiempo en el módulo (General) y también se hará un promedio de bines por lote de los que ha probado (lote).

2.3.3. Test time

En español significa tiempo de pruebas y su nombre es bastante descriptivo, lo que indica es el tiempo que tarda la prueba de una unidad, esto toma importancia tomando en cuenta que bajo algunas condiciones de falla la prueba terminará antes o después. Este dato al igual que los bines general y lote, serán tomados como un promedio de todas las unidades que ha probado en su tiempo dentro de la celda.

2.4. Preprocesamiento de datos

Antes se menciona que dentro del ML los datos lo son todo, y con justa razón pues de ellos dependerá en gran parte el éxito de nuestro modelo de predicción que buscamos realizar. El preprocesamiento de datos e una de los procedimientos más esenciales del descubrimiento de información o KDD (por el inglés knowledge discovery in databases), en estos pasos se suele revisar la integridad , limpieza de datos, además también la transformación y reducción según la convenga para tener una entrada en buen estado para el aprendizaje automatizado. [7]

Capítulo 3

Aprendizaje automático

El flujo de un proyecto de aprendizaje automático está a menudo asociado a un proceso el cuál inicia con la premisa de tener una cantidad considerable de datos disponibles con los cuales trabajar, a esta le llamamos recolección de datos y en el caso de este proyecto los mismos son recolectados automáticamente por parte de Intel y sus datos de test. Posteriormente inicia la etapa de preprocesamiento de datos donde se filtran y se afinan para una detección de patrones más fácil y por tanto obtener un mejor modelo más adelante. Una vez se tiene esto se suele visualizar los datos para ver patrones obvios y tomar decisiones en cuanto al modelo que se quiere usar para pasar luego por la etapa de construcción del modelo que realizará las predicciones. Por último se evalúa la precisión de el modelo conseguido, usualmente los datos disponibles los separamos en dos partes, una para entrenamiento y otra para pruebas, pues se utiliza la parte de pruebas para ver eficacia del modelo y en caso de este proyecto se puede también comparar con lo visto en el mundo real para ver si verdaderamente funciona el modelo. En este proyecto debido a la estructura de los datos, se agrega una etapa extra a la que llamo transformación de datos. Esta tiene el objetivo de traducir los datos de test (que describen rendimiento de procesadores) en datos de colateral que describen el historial de los test colateral.

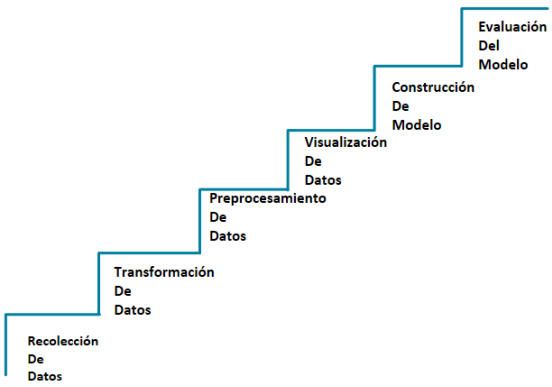

Figura 3.1: Pasos para el proyecto de aprendizaje automático (Creación propia).

3.1. Transformación de datos

Esta etapa del proyecto se puede ver como un paso previo a lo que generalmente se requiere para el aprendizaje automático o bien como parte del preprocesamiento de dato. Es de vital importancia para el proyecto poder obtener características que describan a un colateral para poder introducirlas en un modelo predictivo, los datos brutos que usamos no nos sirven de mucho así como están.

3.1.1. Datos crudos

Los datos con los que cuesto presentan diversas características de las pruebas dirigidas hacia un procesador. estas características vienen en forma de columnas y las filas es cada una de las pruebas en las diferentes máquinas, celdas y unidades. Las columnas que utilicé para dar forma una una nueva caracterización del colateral (*socket* o TIU) son las siguientes.

- VISUAL_ID: Código del procesador, utilizado para contabilizar.
- WITHIN_SESSION_SEQUENCE_NUMBER: Número de secuencia de las pruebas de un mismo procesador (cuando ocupa más de una), se usa para ordenar y localizar fallos sólidos.
- WITHIN_SESSION_LATEST_FLAG: Bandera de última prueba, esto es 'Y' si es la última de las pruebas realizadas a una unidad, utlizado para localizar fallos sólidos.
- INTERFACE_BIN: Código que describe los resultados de las pruebas a una unidad, ya sea éxito o fallo y qué tipo, utilizado para clasificar según fallos recurrentes.
- TESTER_INTERFACE_UNIT_ID: Colateral al cual llamamos TIU por sus siglas, este el número al que se quiere seguirle pista y darle características de rendimiento.
- MODULE: Número de máquina en la cual se realiza la prueba, para este caso tenemos solo 3. Esto lo usamos para ubicarnos espacialmente y no confundir celdas de una u otra máquina.
- SITE_ID: Número de celda dentro de una misma máquina, importante para saber cuál colateral se encuentra dentro de ella en un momento dado.
- TEST_TIME: Tiempo que tarda la prueba realizada a la unidad, utilizada para caracterizar a un colateral.
- DEVICE_END_DATE_TIME: Fecha y hora en la que se realiza la prueba, utilizada para ordenar los datos de forma cronológica.
- LOT: Número de lote (un lote es un conjunto de unidades que comparten características), utilizado para contabilizar y caracterizar un colateral.

Dentro de todos los datos de pruebas estas fueron las columnas seleccionadas para ser procesadas y obtener datos que puedan ser ingresados al modelo de aprendizaje automático.

Se obtienen estos datos de entrada mediante un algoritmo de extracción de información aplicado a las bases de datos de Intel en el cuál se tienen archivos con aproximadamente 4 semanas de datos cada uno y en total casi un año de datos de un producto en específico. Estas en general tienen un rango temporal entre enero del 2021 y mayo del 2022, además suman 825MB de almacenamiento.

3.1.2. Datos transformados

14horas Para la transformación de datos utilizamos la librería de pandas para ayudarnos con el manejo de arreglos y facilitar el proceso con sus diversas funciones. Se crea el archivo llamado Data_Transformation.py donde inicialmente deben ser importados los datos (estos se introducen al algoritmos mediante argumentos).

Uno de los grandes retos de este proyecto fue el de sacar de la ecuación los fallos sólidos de la ecuación, como se explica en la sección de teoría, estos son varias pruebas que se le realizan a una unidad que obtienen el mismo resultado confirmando que efectivamente es un defecto que tiene el procesador y no debido a alguna anomalía en la máquina de pruebas, si fuera debido al instrumento de pruebas veríamos cambios entre las pruebas que se realizan. Para esto la estrategia tomada fue ordenar los datos en primera instancia por unidad y luego por el número de secuencia que nos dejará recorrer el arreglo en el orden en que se van realizando las pruebas y en conjunto con con la bandera de última prueba se identifica cuando el fin de las pruebas hacia la unidad. Dicho esto, en esencia el algoritmo recorre todos los datos y se fija que si todas las pruebas hacia una unidad reflejan el mismo resultado, lo agrega a una lista llamada Solid_visual para que luego sean ignorados en etapas posteriores. Esto quiere decir que los resultados que obtendremos son solo las unidades que tuvieron varias pruebas con diferentes resultados que no está ni cerca de ser la mayoría por lo que deberá ser procesado una cantidad considerable de datos. Como se debe recorrer todo el arreglo de datos esto significa un atraso grande de tiempo de ejecución, sin embargo es la única manera de hacerlo porque en etapas posteriores se deberá recorrer el arreglo pero ordenado por máquina, por celda y por tiempo, de esta forma no es posible ver esos fallos sólidos ya que una unidad no realiza dos pruebas en una misma celda sino que lo hace en diferente y al mismo tiempo que se prueban otras en otra celda y por resto se perdería visibilidad.

Una vez tenemos identificadas la unidades sólidas se hace un barrido de los datos por máquina y por celda a los datos que fueron previamente ordenados por fecha para saber el orden de los hechos. Una parte interesante de esta etapa es la manera de como vamos a clasificar un colateral como marginal ya que en los datos tampoco se va a indicar explícitamente. Para cubrir este problema trabajamos bajo el supuesto que en una celda un colateral va a trabajar hasta que su comportamiento sea malo y deba reemplazarse por otro en buen estado. Bajo esta premisa nos permitimos hacer una recuento característico del histórico de este colateral desde que es puesto en la celda hasta que falla y debe ser reemplazado. Claro que esta idea es algo simplista y tiene sus limitaciones ya que lo que ocurre en la realidad no es completamente eso todo el tiempo ya que pueden ocurrir cambios esporádicos por algunas situaciones externas a la marginalidad, sin embargo se trabaja con este supuesto importante a tomar en cuenta.

Dicho esto el programa recorre los datos como se menciona antes y recolecta características del colateral que se encuentra en la celda actualmente para conseguir una tabla con las siguientes características.

- Socketing: cantidad de accionamientos o pruebas que lleva desde que fue puesto en la celda.
- TIU: Número de colateral, TIU en este caso.
- G/B_flag: Aquí codificamos como 0 si es un dato de un colateral defectuoso o 1 si es un colateral en buen estado.
- Test_Time: Promedio de tiempo que duran las pruebas en este colateral.
- Bines_General: Cantidad de bines de fallo que ha tenido.
- Bines_NLot: Cantidad de bines de fallo por lote, como se pruebas varios lotes, se toma un promedio para este caso.
- #: Luego se viene una serie de columnas con números de 1-99 que representan los bines de fallo de los cuales se tendrá registro si ocurren.

Una vez tenemos esta información sobre los colaterales malos se debe incluir datos que reflejen características de un colateral bueno que básicamente es todo lo demás pero no es recomendable introducir al algoritmo una relación de datos buenos y malos tan desproporcionada como podría ser 1500 datos buenos y uno malo. Para resolver este problema se crea una relación de proporcionalidad que se llama “balance” la cual es también una entrada por argumentos y nos indicará por ejemplo que si es 5, existirán 5 datos buenos por cada dato malo. Este es un parámetro más que viene a tomar partido en este proyecto. Para las pruebas iniciales este se toma como 5 sobretodo para reducir un poco el tiempo de ejecución que se puede elevar bastante con facilidad.

Una vez tenemos todo esto listo, se realiza un filtrado previo a estos datos para no meter tanto ruido, a esto se eliminan todas las columnas de bines que no se usaron (por lo tanto todos los datos tienen cero en ellas), también se eliminan colaterales con *socketing* cero o que no tuvieron ningún bin de fallo en todas sus pruebas, y finalmente las filas que están vacías y hacen ruido. Este resultado se escribe en un archivo de excel indicado en las entradas de argumentos cuando se corre el programa de transformación de datos.

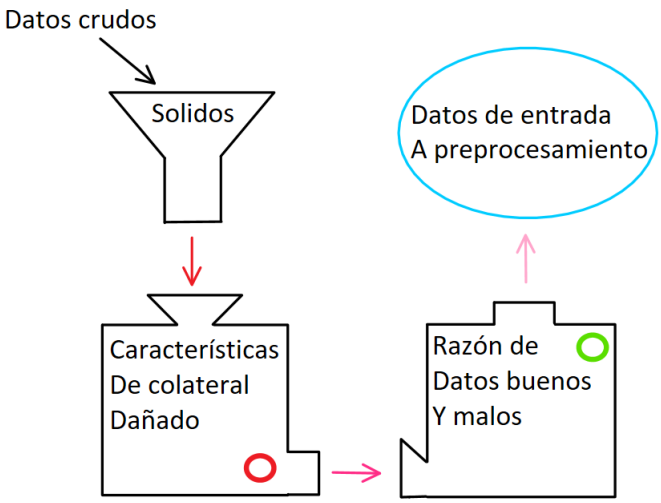

Figura 3.2: Flujo simplificado para la transformación de datos(Creación propia).

3.2. Preprocesamiento de datos

El siguiente paso en la preparación de nuestros datos es el preprocesamiento en donde se filtrará y dará forma al arreglo de datos con el que entrenaremos nuestro modelo de aprendizaje automático. Es importante mencionar que el proceso transformación que se describe anteriormente debe ser ejecutado varias veces a varios archivos para poder obtener una cantidad de datos funcional. Para esto de utiliza un algoritmo aparte llamado “Master.py” el cual realiza llama a la transformación de datos de varios archivos locales (previamente descargados de la base de datos de Intel). La salida de este algoritmo son varios archivos con información, específicamente 14 que usaremos para alimentar nuestro algoritmo de aprendizaje automático.

3.2.1. Integración de datos

En este paso se busca la mezcla de todos los datos en archivos dispersos por el paso anterior dentro uno solo para poder manejar todo de la misma manera y en conjunto. Para esto se utiliza la ayuda de la librería pandas para importar todos los datos de los archivos y posteriormente ejecutar la función de concatenar disponible en la misma librería para juntar todo lo que tenemos, al ejecutar esta acción tendremos aproximadamente 58000 líneas de datos.

3.2.2. Limpieza de datos

Si bien es en un inicio se ponen todos los atributos disponibles para caracterizar un colateral, no todos ellos son útiles e incluso pueden provocar ineficiencia en el entrenamiento si no están debidamente

filtrados. Con este paso se hace parte de esto, inicialmente buscamos los valores faltantes (denotados como NaN en el arreglo de datos), estos son columnas que a la hora de mezclar los datos en el paso anterior quedan sin información, y si están sin información es debido a que no tenían datos de ese bin así que se rellenan esos espacios con cero.

Aparte de datos en blanco también es posible que tengamos ruido dentro de nuestros datos que pueden ser debido a bines muy esporádicos en un colateral o una falla que no este asociada a este colateral. Por esta razón se decide hacer un filtrado de valores mínimos dentro de las columnas de bines, se da el criterio de 5 fallos de un bin entonces si hay alguna columna que no posea al menos una fila con un valor mayor a 5 será eliminada de nuestra ecuación ya que se toma como ruido. De lo anterior hablamos acerca de las columnas de bines pero hay una quizá un poco más importante que es la de "Socketing" ya que una cantidad muy baja de test dento de un colateral nos dará datos poco consistentes y probablemente que se salgan de nuestro objetivo ya que estos pueden deberse a malas reparaciones, cambios de colaterales entre celda o alguna otra investigación de ingeniería, por este motivo se toma el criterio de haber probado al menos 30 unidades, sino esa fila no será tomada en consideración.

Para finalizar esta etapa hay dos columnas más que no nos interesan de momento que son la del nombre del TIU y la del bin 1, este último debido a que ese ya cuenta como un test bueno y queremos fijarnos en cuales son los patrones de falla que tiene el colateral en su marginalidad por tanto estas son eliminadas.

3.2.3. Reducción dimensionalidad

La Reducción de dimensionalidad tiene el objetivo de reducir la cantidad de información con la que es entrenado el algoritmo de aprendizaje automático, y así con seguir los mismos resultados con la menor dimensión posible, en otras palabras buscamos seleccionar los datos que nos aporten información relevante para la detección de marginalidades. En este sentido al tratarse de un algoritmo que funciona mediante la unión de distintas probabilidades que asumimos como independientes entre sí, buscamos que los datos estén lo menos relacionados posibles sin dejar de lado el sentido de importancia que se le da en indicadores de marginalidad. El método que se utilizó como criterio para eliminar atributos es el de correlación, la librería pandas cuenta con una función llamada "cor" la cual nos muestra la matriz de correlación entre todas las variables, esto nos deja saber que cuales tienen una mayor o mejor correlación en donde 0 es nada relacionadas y 1 es totalmente relacionadas, así como también relaciones inversas representadas por números negativos.

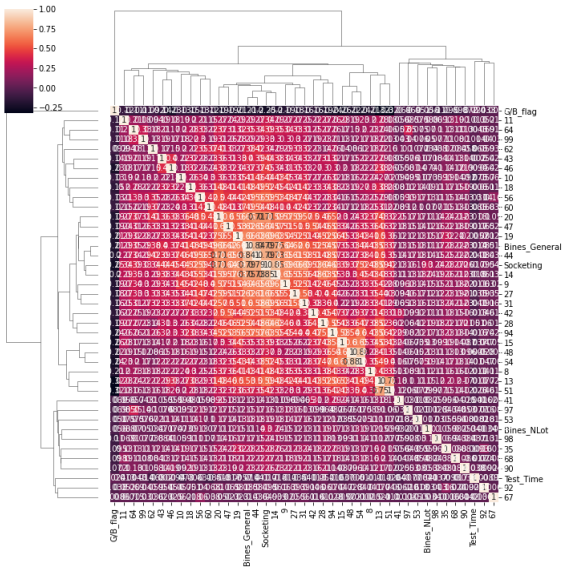

Figura 3.3: Mapa de calor acerca de la correlación de las variables. (Creación propia).

Con ayuda de la librería de seaborn es posible obtener un gráfico de calor que describa la relación de todas estas variables entre sí así como se muestra en la figura 3.3. Aquí cabe resaltar que los colores claros (más cercanos al blanco) son las relaciones más fuertes y como es de esperarse, el socketing se relaciona fuermenente con varios bines ya que este representa desgaste en el colateral, por este motivo no se remueve. Sin embargo al ser una cantidad de variables bastante alta se opta por ver estos datos en excel para resaltar condicionalmente las correlaciones fuertes y tomar decisión de qué es lo mejor que se puede hacer con ellas. Para este criterio se toma en cuenta que una relación mayor al 70 % es fuerte. Con esto tenemos los siguientes hallazgos.

1. Bin 51 y bin13: 0.75
2. Bin 44 y bin 14: 0.73
3. Bin 44 y Bin 20: 0.71
4. Bin54 y Bin48: 0.88

La relación 1 es un comportamiento sumamente esperado ya que ambos bines es bien sabido en el área que están relacionados con problemas en su mayoría por problemas asociados al socket donde se pone la unidad que se prueba, si bien ambos tienen la misma procedencia, no tienen la misma causa raíz por lo que sin importar que se den con una frecuencia similar. se decide dejar ambos como parte del análisis.

En cuanto a la relación 2 y 3 vemos que son bien representadas por la el bin 44 por lo que decidimos dejar este como representante y olvidarnos por ahora del bin 14 y 20. En la relación 4 se deja por experiencia en bin54 ya que al igual que antes, este está usualmente asociado a un fallo de TIU.

3.2.4. Normalización de datos

La normalización es una de las técnicas más utilizadas en el preprocesamiento de datos, este es el proceso de cambiar la escala y rangos de los datos. Con este paso buscamos no crear algún sesgo al algoritmo con respecto a las diferencias de magnitud y variación que existen entre un atributo u otro. Para esto se utiliza la librería de preprocesamiento que ofrece sklearn en donde tenemos la función de normalización por "StandardScaler" el cual nos ayudará de una manera sencilla a normalizar los datos forzando los promedios de cada atributo a cero y la varianza unitaria. Para este caso no es necesario restringir el rango de datos a 0 y 1 como suele hacerse en otros algoritmos ya que el modelo que buscamos no es tan sensible a estos cambios.

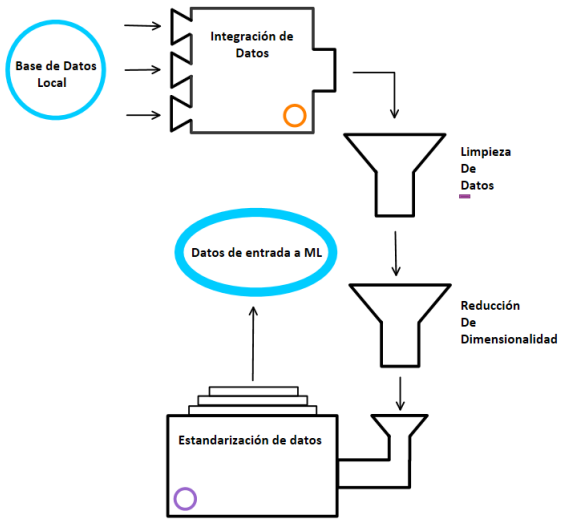

Figura 3.4: Flujo simplificado para el preprocesamiento de datos (Creación propia).

3.3. Visualización de Datos

Un paso importante para la toma de decisiones en cuanto al modelo que se usará para predecir datos ya que de esta manera podemos observar tendencias entre las relaciones de las distintas variables, por ejemplo observar si hay algún comportamiento lineal, de segundo orden o agrupaciones al rededor de ciertos valores.

Al ser un arreglo con múltiples características del colateral nos formaría un gráfico multidimensional que complica la visualización. Para esto se usan distintas estrategia en las cuales se nos permite tener una idea del comportamiento de las variables. La primera de ellas es la opción ofrecida por seaborn en la función "pairplot" la cual facilita la creación de graficos de todas las relaciones existentes en donde se grafican todas contra todas las variable. Si bien esta es una herramienta sumamente útil

y fácil de utilizar, la cantidad de variables que se usan demanda demasiada memoria RAM para poder utilizarlo y cuando se intentaba no lograba concretarse, por tanto se seleccionan solo ciertas variables que deben ser las más significativas. Con criterio sostenido por la experiencia en el área, decido graficar el Socketing, Test_time, Bines_Nlot, bin8, bin 13, bin15 y bin51 los cuales están asociados a fallas en esta parte.

En la figura 3.6 podemos ver los resultados de esta prueba que se muestra en forma de arreglo para poder ver la relación con dada una de las variables, además se muestran como puntos de color naranja los datos de colaterales en buen estado y el azul los que están en buen estado. Se pueden destacar las distribuciones que se encuentran en la diagonal que muestran la relación de una variable contra si misma, aquí es fácil observar la relación que existe entre los colaterales buenos y malos dentro de una misma variable, es interesante que no se observa ninguna variación en la media sino solo en la magnitud de la media, ya que son mucho menos, exceptuando el tiempo de prueba que como es esperado, es un poco más corto en promedio que una unidad en buen estado ya que si se encuentra una falla, la prueba termina. Por lo demás sí podemos ver tendencia a agruparse en distribuciones gauseanas o agrupadas al rededor de un solo valor pero de igual forma siguiendo entremezcladas los datos buenos como los malos.

En complejo para la mente humana ya empezar a pensar como interaccionas todas estas relaciones entre ellas mismas, no solo entre dos variables, sino que todas juntas a la misma vez ya que son muchas interacciones pasando al mismo tiempo. Hay una manera de intentar representar y visualizar todas estas en un mismo gráfico de dos dimensiones. Este es el otro método que se estudia para observar si existen tendencias en los datos.

T-distributed Stochastic Neighbor Embedding o TSNE por sus siglas en inglés es una herramienta utilizada para visualizar datos de altas dimensiones, este es parte de lo que ofrece la librería sklearn e intenta convertir similitudes en los puntos de los datos en probabilidades de unión con el objetivo de minimizar y de cierta forma proyectar estos puntos en múltiples dimensiones a una dimensión menor.

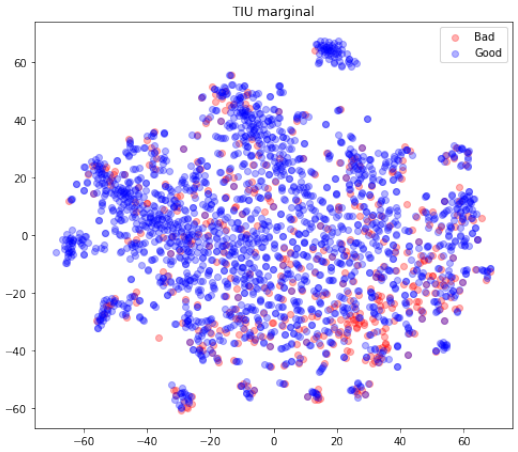

Figura 3.5: Visualización de todas las variables TSNE (Creación propia).

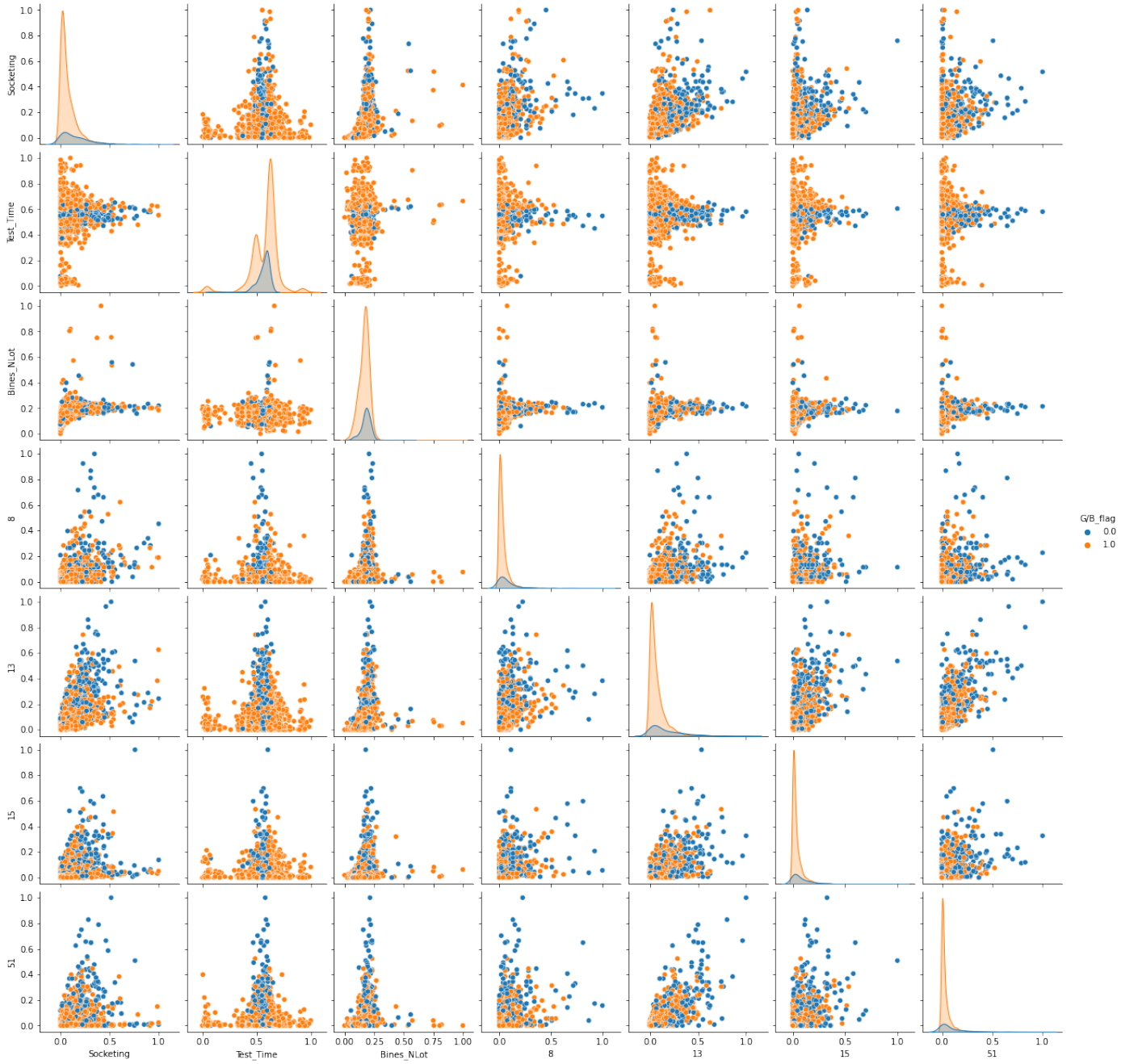

Figura 3.6: Visualización de pares (Creación propia).

En la figura 3.5 vemos los resultados de TSNE los cuales tienen el código de colores de azul para las buenas y rojo para los datos malos. Estos datos fueron previamente normalizados para que sea comparables con respecto a las demás variables. Si bien observamos algunas agrupaciones pequeñas aisladas y algunos puntos con mayor concentraciones de datos, siempre se ve que todos ellos tienen contenido tanto colaterales buenos como malos y por tanto no se ven tendencias entre ellas con esta simplificación. Ya que no vemos estas tendencias podemos descartar modelos como el de regresión lineal o clustering, se decide usar para este proyecto el modelo de *Naive Bayes* debido a su capacidad de clasificación de datos de múltiples características y con una implementación relativamente sencilla que podemos obtener buenos resultados.

3.4. Construcción de modelo

Ya definido el algoritmo de aprendizaje automático que será utilizado el cual será Naive Bayes, ahora se debe decidir entre los distintas variantes del modelo que ofrece sklearn. Las tres que evaluaremos tienen distintas características, "Multinomial" es un modelo enfocado en datos discretos en donde aparecen cuentas sobre características y se muestra de cierta forma una frecuencia, los datos deben preferiblemente tener una distribución multinomial como la que observamos en los bines por lotes, además también todos los datos de bines son datos discretos por lo que es posible que este nos sea de utilidad.

Otra variante es la "Gaussian" la cual como su nombre lo dice, espera una distribución gaussiana de las variables así como la que observamos y mencionamos de la figura 3.6, además es más adecuada para datos continuos como bien puede ser la temperatura (que podría ser utilizada como dato la pieza de la máquina llamada *thermal head*). Por útilmo está la opción de bernuilli que tiene un mayor enfoque hacia datos binarios por ejemplo, si las TIU fueron reparadas en las ultimas 48 horas (sí o no), debido a esto es esperable que esta tenga un rendimiento más bajo que los otros modelos.

Para la construcción de estos modelos es importante recordar que la base de todo esto es el teorema de Bayes el cual está descrito en la ecuación 2.1, la forma de aplicar esto en un algoritmo de clasificación de muchos atributos es inicialmente en nuestro caso calcular las probabilidades de que cada uno nuestros atributos aparezcan en un colateral bueno o en uno malo, por ejemplo calcularemos la probabilidad de que un colateral bueno tenga un bin13 denotado por la $P(bin13|TIUbuena)$, esto debería hacerse para todos los atributos y cada una de las posibilidades de clasificación (bueno o malo). Además de esto deberíamos calcular la probabilidad de que un colateral sea malo y la probabilidad de que sea bueno, a esto se le llama estimación inicial y es con lo que jugamos en la transformación de datos en cuanto a la relación que existe entre cantidad de datos con colaterales bueno y malos.

Ahora que tenemos estas probabilidades calculadas se hace un calculo de probabilidad para un dato en concreto que nos ayudará a determinar si este es parte de los bueno o de los malo, la forma en la que se hace esto es simplemente una multiplicación de probabilidades de los atributos con la estimación inicial, por ejemplo:

$$P(TIUBuena) = P(Buena) \cdot P(socketing|Buenas) \cdot P(Bin13|Buena) \cdot ...$$

Lo anterior debe hacerse también para la $P(TIU mala)$ y así tener un punto de comparación, la probabilidad más alta es la que será tomada como verdadera y la que dictará la clasificación. Lo expliqué anteriormente es bastante general de la manera que se aplica este método ya que hay muchas optimizaciones que deben ser tomadas en cuenta por ejemplo cuando uno de los atributos es cero la probabilidad será cero y se puede estar tomando una asunción errónea, para esto hay métodos de como tratar estos casos como el de agregar un 1 a todos los datos para que sea proporcional pero que la probabilidad sea distinto de cero. También cuando una probabilidad es extremadamente pequeña nos afectará en nuestra decisión final por lo que a menudo suele tomarse una normalización logarítmica de estos datos para hacer la comparación.

Afortunadamente existen librerías de python que facilitan todo este proceso, para resolver estas operaciones y hacer más optimizaciones utilizamos la librería sklearn que tiene una sección de modelos *Naive Bayes* donde podemos importar las variaciones de este modelo antes mencionadas. La forma en la que se debe aplicar este modelo es muy sencilla, solo se crea un objeto del tipo del modelo que queramos crear, por ejemplo "MultinomialNB" que es como sale en la librería y este objeto tendrá varias funciones, utilizamos la que se llama "fit" y enviar como argumentos nuestros datos de entrenamiento y las etiquetas de cada uno de los datos para que tengamos un objeto utilizable para hacer predicciones, métricas, entre otros.

No es viable solo entrenar un modelo e irnos en primeras a probarlo con datos sino que debemos medir su rendimiento. Se suele partir los datos disponibles para el modelos en dos partes, una para entrenar el modelo y otra para hacer predicciones y medir que tan certera es la respuesta (ya que los datos que predecimos ya tienen su clasificación previa). El modelo que ya tenemos entrenado tiene varias funciones para medir el rendimiento de las cuales estaremos utilizando las siguientes cuatro.

- score: Esta es de las más importantes y nos indica la relación de las muestras predichas de forma correcta y las que no.
- recall_score: Mide la relación de error causado por falsos negativos, en otras palabras, muestras clasificadas como malas pero que están buenas.
- precision_score: Mide la relación de error causado por falsos positivos, en otras palabras, muestras clasificadas como buenas pero que están malas.
- f1_score: Es una relación entre falsos positivos y falsos negativos por lo que nos resume un poco de las dos anteriores.

Estos indicadores nos darán un mejor panorama de cual de estos es el más apropiado para utilizar en estos datos y detectar la marginalidad de un TIU.

Con esto en mente aun hace falta tomar en cuenta que estamos partiendo nuestros datos en partes y el resultado del modelo estará influenciado por los datos con los cuales se está entrenando, así que este va a depender de cuál de los fragmentos va a ser utilizado para entrenarse. Con el objetivo de evitar este sesgo creado por cuales datos son los utilizados para que bloque, se aplica una técnica llamada cross-validation o validación cruzada en español.

Esta es una técnica en la cual es se parten todos los datos disponibles en varios pedazos, por ejemplo en 3 y se utilizan el 1 y 2 para entrenar y 3 para probar, esto se hace con todas las combinaciones posibles, se mide rendimiento y se realiza un promedio entre todos para tener una idea más clara de que es lo que veríamos en una aplicación real del modelo. Para automatizar este proceso de partir los datos se utiliza una librería de sklean que se llama model_selection la cual nos facilita la función "KFold" que se encarga de devolvernos los índices de las particiones creadas y así poder tomar acción con respecto a eso, para estas pruebas se utilizan 10 particiones ya que es la cantidad que recomiendan distintos autores.

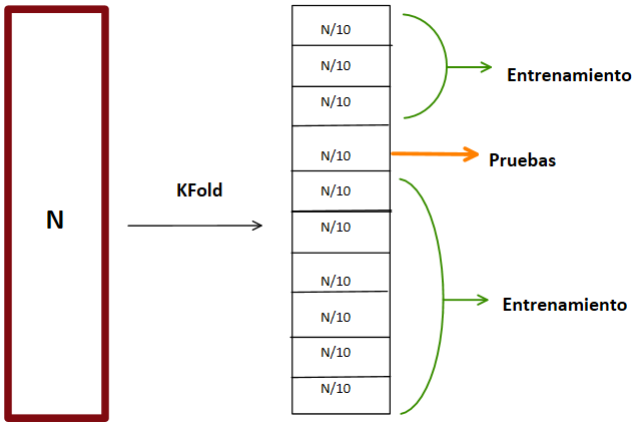

Figura 3.7: Validación cruzada (Creación propia).

3.4.1. Repositorio Git con el código

https://github.com/felipebm09/Proyecto-Electrico/

3.5. Aplicación del modelo

Para finalizar se programa un script de python adicional el cual nos permite evaluar nuevos datos aplicando el modelo escogido para poder predecir cual o cuales colaterales deben ser revisados lo más pronto posible.

Para se hace importando los nuevos datos y debemos tomar pasos muy similares a las etapas de transformación y preprocesamiento, incluso buena parte del código es reutilizado. En cuanto a la transformación de los datos, la diferencia es que ahora no solo necesitamos registro del último TIU que fue puesto en la celda entonces agregamos una columna de maquina y celda para poder localizarlo espacialmente vamos iterando los datos como se hizo en transformación pero solo se guardan los últimos colaterales, recordando que esta vez no se clasifican (esto lo hará el modelo) sino que solo ocupamos datos característicos.

Para el preprocesamiento, ya sabemos cuales son las columnas que fueron dejadas para entrenar el modelo (no podemos predecir con columnas nuevas a menos que reentrenemos) por lo tanto eliminamos todas las columnas que no se encuentran ahí y reordenamos tal y como está en los datos de entrenamiento. La parte de normalización es un poco distinta ya que debemos normalizar estos datos de la misma forma en que fueron normalizados los de prueba, entonces para satisfacer esta necesidad en el archivo de preprocesamiento se incluye la librería pickle que nos ofrece la función de "dump" para poder guardar los datos de normalización, posteriormente se importan con la función de "load" que ofrece pickle y pueden ser utilizados para normalizar los datos actuales.

Una vez tenemos nuestros datos listos, al igual que se hizo con la normalización, los datos del modelo deben ser guardados en un archivo aparte para importarlos y poder utilizar el modelo en los nuevos datos. Una vez importados solo hace falta utilizar la función "predict" del modelo para tener las respuestas de clasificación.

Cuando tenemos las respuestas llega la hora de visualizarlas de una manera comprensible para el usuario entonces se parten los datos según máquina y celda, con ayuda de la librería matplotlib se grafican los datos como un arreglo de colores que resaltará las celdas de la máquina que deben ser revisadas.

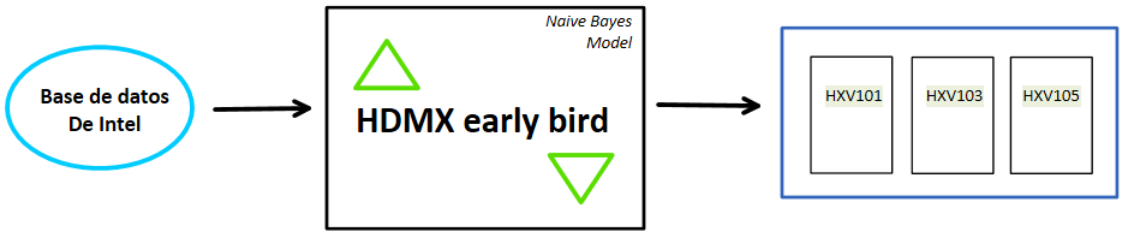

Figura 3.8: Flujo del producto final (Creación propia).

Capítulo 4

Análisis de resultados

Se analizarán los resultados que se fueron obteniendo en cada etapa del proyecto para ir dando un hilo conductor hacia el resultado final. Empezando por la etapa de transformación de datos, pasamos de una base de datos basado en descripción de pruebas a un procesador hacia una descriptiva de cada colateral como se muestra a continuación

Socketing	TIU	G/B_flag	Test_Time	nes_Gene	Bines_NLo	1	8	9	10	11	13	14	15	18	19
795	P4003	1	173.3087	63	31.5	39	1	0	0	0	0	0	0	0	0
1590	P4003	1	136.5504	141	47	95	1	0	0	0	2	1	0	0	0
2385	P4003	1	170.6388	206	51.5	135	1	1	0	0	2	3	0	1	0
3181	P4003	1	172.1662	310	51.667	208	1	1	0	1	3	3	0	1	1
3976	P4003	1	190.3632	426	53.25	299	2	1	0	1	4	3	0	3	2
5567	P4003	0	173.65	592	53.818	409	2	2	0	1	8	6	1	3	3

Figura 4.1: Resultados en la transformación de datos.

Podemos observar que en esta ahora tenemos filas que tienen asociado un TIU, las veces que ha hecho pruebas, columnas por bines, entre otras características y quizá la más importante es la de "G/B_flag" que es nuestra etiqueta hacia esa fila, cabe resaltar que hay tanto filas con colaterales buenos como malos, hay que tomar en cuenta que en esa figura no se muestran todas las columnas de bines. Aquí se cumple perfectamente el objetivo y tenemos datos útiles para etapas posteriores.

Ahora bien la siguiente etapa es la de preprocesamiento de datos en donde dejaremos solo los datos que nos sean útiles. Se integran todos los archivos que salen de la etapa anterior y se remueven las columnas de "TIU", bin 1 y las tomadas de resultados de correlación, aparte también todas las filas que tengan socketing menor a 30 y las columnas que no tengan al menos un dato mayor a 5. Posterior a esto nos quedamos con un arreglo de 35 columnas y 2468 filas que serán las responsables de entrenar nuestro modelo. En esta etapa también se visualizan los datos con las gráficas mostradas en las figuras 3.6 y 3.5.

Llega la hora de hacer y evaluar nuestro modelos con los datos que ya están listos y como se mencionó en la sección anterior, se utilizarán 3 indicadores de rendimiento como referencia para elegir cuál es la variante del modelo Naive Bayes más conveniente a utilizar.

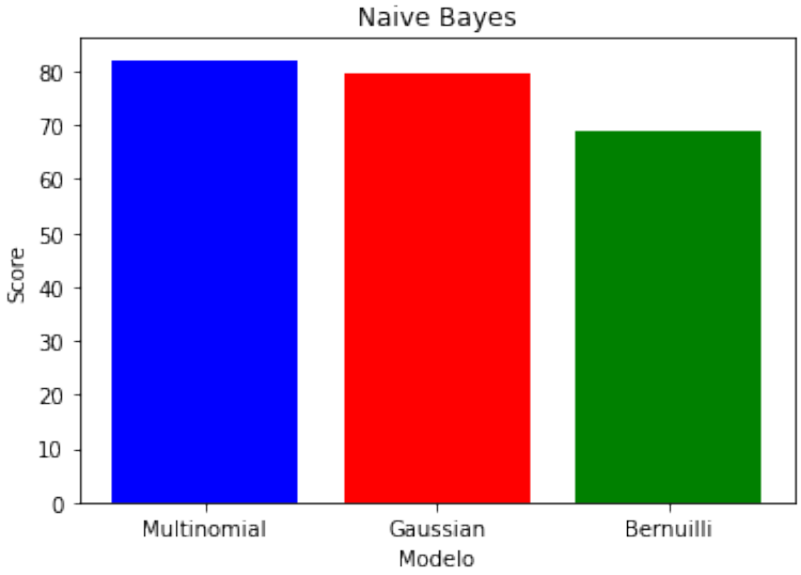

Figura 4.2: Resultados indicador score.

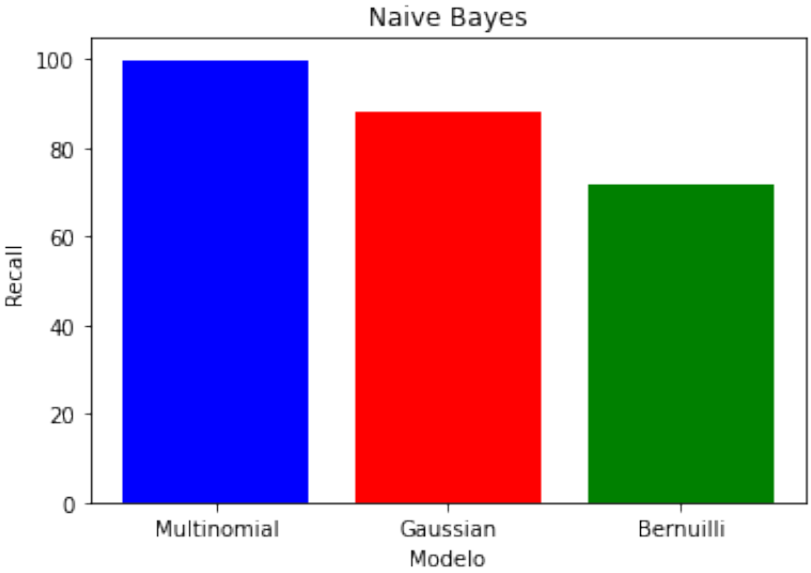

Figura 4.3: Resultados indicador recall_score.

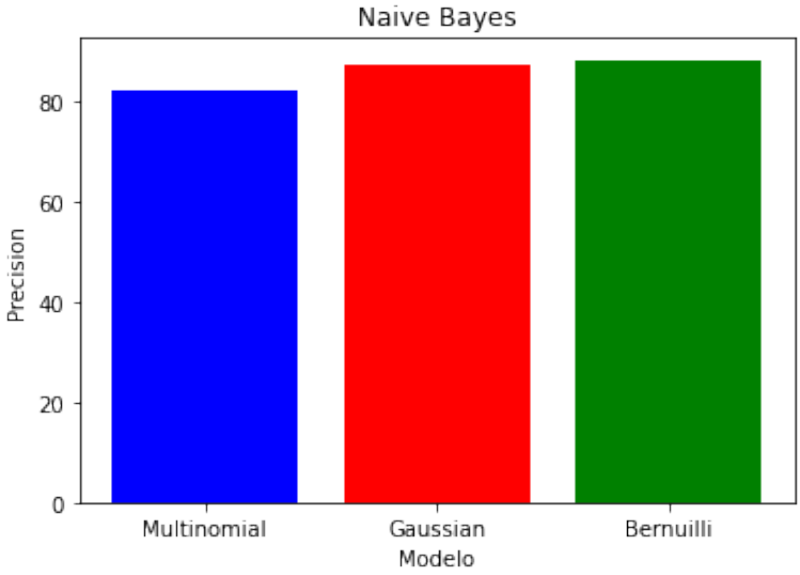

Figura 4.4: Resultados indicador precision_score.

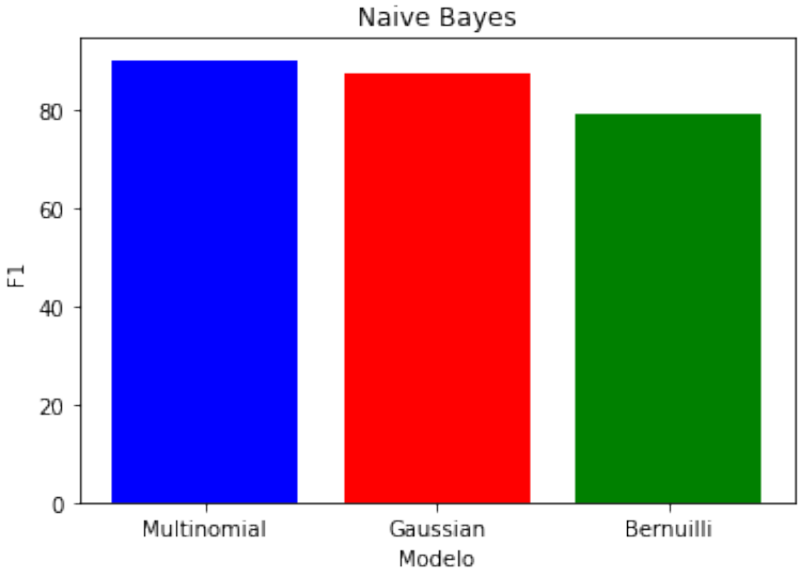

Figura 4.5: Resultados indicador f1_score.

Analizando las figuras 4.2, 4.3, 4.4 y 4.5 podemos darnos cuenta que el que tiene un mejor rendimiento prediciendo de marginalidades de manera correcta el multinomial que tiene un rendimiento de un 82.1 % seguido del gaussian con un 79.5 % y por último el bernuilli con un 69.1 %. En cuanto al recall el multinomial vuelve a ser el mejor con un 99.8 %. En precision el ganador aquí es bernuilli con un 88.1 % y por último nuestro indicador de f1 es ganado por el multnomial que obtuvo un 90 %.

En los datos anteriores se puede inducir que el mejor modelo en definitiva es el multinomial y el peor bernuilli como habíamos esperado en un inicio. Los rendimiento siguen estando bajos sin llegar a un 90 % en score lo cual es un posible motivo para replantear en un futuro las evaluaciones de algoritmos de clasificación distintos a Naive Bayes sin embargo son resultados aceptables para utilizar como una herramienta de alerta inicial donde nos puede indicar donde buscar y corroborar esta información complementándola con otras herramientas de análisis ya implementadas en el proceso de Intel.

Dicho lo anterior se decide que el mejor algoritmo de Naive Bayes que puede ser utilizado el multinomial y se procede con una prueba con datos completamente nuevos de la presente semana en donde se nos indica cuales son las celdas que deben ser revisadas obteniendo el siguiente resultado gráfico.

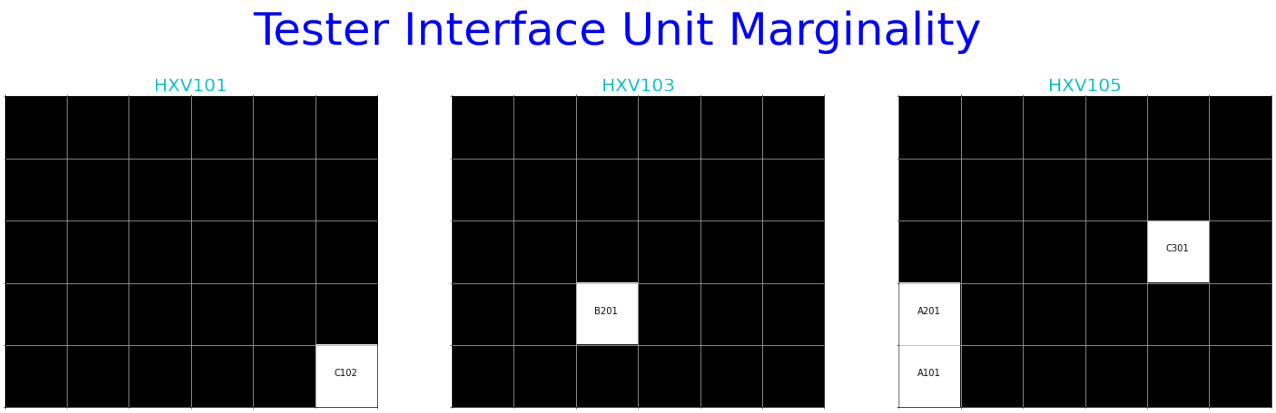

Figura 4.6: Resultado con nuevos datos.

Como se observa en la figura 4.6 nos indica las celdas que hay que revisar en color blanco y a la misma vez se revela el nombre de la celda. Debido al reducido tiempo que queda para pruebas físicas en este proyecto se hace una corroboración de rendimiento con otras herramientas para observar que tan veraz es esta información que nos presenta el modelo.

Tenemos 4 celdas que revisar, indagando más afondo en la actualidad de esos colaterales se descubre que tanto en la máquina HXV101 celda C201 como en la HXV103 celda B201 las TIU tuvieron problemas y ya no se encontraban ahí sino que están en proceso de reparación, por esta razón se toman estas como predicciones certeras.

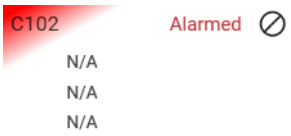

Figura 4.7: Resultado real máquina 101.

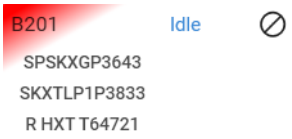

Figura 4.8: Resultado real máquina 103.

En cuanto a la máquina 105, sí tenía todas sus celdas haciendo pruebas y por lo tanto se hace un análisis manual de rendimiento para determinar si son candidatas a una revisión.

C301	
BB2TXH1P3651	**100.00%**
1	92.86%
94	1.10%
54	0.66%
31	0.66%
13	0.50%
14	0.45%
28	0.39%
44	0.37%
9	0.37%
56	0.29%

A101	
BB2TXH1P4009	**100.00%**
1	86.94%
13	3.30%
51	2.68%
94	1.38%
54	0.85%
46	0.62%
44	0.53%
31	0.53%
8	0.48%
14	0.44%

A201	
BB2TXH1P3822	**100.00%**
1	89.32%
31	2.65%
94	1.28%
44	1.00%
13	0.88%
54	0.79%
14	0.47%
56	0.44%
9	0.33%
28	0.28%

Figura 4.9: Resultado real máquina 105.

En la figura 4.9 podemos observar un top 10 de resultados de rendimiento de los tres colaterales en donde podemos ver que la celda C301 tiene un 92.86 % de testeos con resultados positivos y la A201 un 89.32 % lo cual está dentro de lo normal ya que en la práctica un 90 % de rendimiento es aceptable por distintas razones del proceso. Sin embargo la celda A101 sí tiene un rendimiento más bajo y además un 3.3 % de bin 13 lo cual está alto con respecto a la media normal que es por debajo de aproximadamente

un 1.5 %, por esta razón este sí sería un candidato a revisión e inclusive es probable que pronto se lleve a revisar por parte de otro indicador interno de Intel. Lo antes dicho nos deja con un total de 3 predicciones correctas y 2 incorrectas que es un 60 % de rendimiento, esto no está mal si se utiliza en conjunto con otra herramienta, como se mencionó antes, puede ser utilizada como indicador de sospecha. Este 60 % del que se habla sería de la información más relevante para este objetivo sin embargo si se quisiera medir exactamente cual es su rendimiento tendríamos que evaluar todas y cada una de las celdas para sabar si las que está diciendo que están buenas realmente lo están.

Capítulo 5

Conclusiones y recomendaciones

Los resultados del proyecto no son fallidos y se logra cumplir con todos los objetivos del proyecto exceptuando la revisión física de estos colaterales, sin embargo el rendimiento está por debajo de lo esperado y esto es motivo para replantearse los modelos o también reajustar los parámetros que tenemos para que este proyecto funcione.

Hay que tomar en cuenta que este proyecto tiene diversos supuestos que podrían se motivo de una baja en el rendimiento. Se toma en cuenta que los colaterales cuando son cambiados en el historial de una celda es porque están malos lo cual no es necesariamente eso lo que sucede todo el tiempo, además que tenemos un parámetro de balance entre datos de colaterales buenos y datos de colaterales malos que fue tomado arbitrariamente y no fue posible experimentar con él debido al tiempo que conlleva generar los datos.

El algoritmo de transformación de datos tiene varias oportunidades de mejora en cuanto a los supuestos de que un colateral este malo, donde habría que recurrir a otra fuente de información adicional nueva o ya existente y además también en el tiempo de ejecución ya que lleva mucho tiempo recorrer las cantidades tan grandes de datos y además este proceso se realiza en múltiples ocasiones dentro del mismo algoritmo lo cual podría mejorarse.

Debe evaluarse la posibilidad de incorporar otros datos significativos acerca de los colaterales como por ejemplo para las TIU existe un contador interno que nos puede dar información de qué tan viejo y por tanto que tanto desgaste puede tener.

Apéndice A

Código Transformación de datos

```
# %% [markdown]
# ## Proyecto El ctrico - HDMX early bird
# ### Autor: Felipe Badilla Marchena - B70848
# ## agvs: 1: Archivo a leer, 1: Archivo donde escribir resultado,
    3: Relaci n de proporci n

# %% [markdown]
# #### Preprocesamiento de datos:

# %%
# Librer as a utilizar

import pandas as pd
import numpy as np
import os
import sys
print('Running preprocessing...')

# %%
# Manejo de archivos
file_name = str(sys.argv[1])

excel_filename = file_name
print('Analyzing document: ',excel_filename)

# Se importan los datos
data = pd.read_excel(excel_filename)

df = pd.DataFrame(data, columns= ['VISUAL_ID', '
    WITHIN_SESSION_SEQUENCE_NUMBER',
```

```
29                                  'WITHIN_SESSION_LATEST_FLAG', '
                                       INTERFACE_BIN',
30                                  'TESTER_INTERFACE_UNIT_ID', '
                                       THERMAL_HEAD_ID',
31                                  'DEVICE_TESTER_ID', 'MODULE', '
                                       SITE_ID', 'TEST_TIME',
32                                  'DEVICE_END_DATE_TIME', 'LOT'])
33 # Ordenamos por fecha y hora
34 df = df.sort_values(by=['DEVICE_END_DATE_TIME'])
35 df = df.reset_index(drop=True)
36 df.shape
37
38
39 # %% [markdown]
40 # #### Filtrado de bines s lidos y conteo de bines por colateral
       general y por lote
41
42 # %%
43 # Recorremos el array para ver s lidos y los eliminaremos de la
       lista (lista nueva llamada df_Switching)
44
45 # Solid_index = []
46 Solid_visual = []
47 SV1 = []
48 SV3 = []
49 SV5 = []
50
51 Current_retest_index = []
52
53 df = df.sort_values(by=['VISUAL_ID','WITHIN_SESSION_SEQUENCE_NUMBER
       ']) # Ordenamos por visual y n mero de secuencia
54 df_Switching = df.copy()
55
56 diferent_flag = 0
57 bin_switch_flag = 0
58
59 prev_visual = ''
60 prev_bin = 0
61 current_bin = 0
62 current_visual = ''
63
64 total_units = 0
```

```
65
66 print('Buscando␣unidades␣s lidas!')
67 # Recorremos los datos
68 for index, row in df.iterrows():
69     prev_bin = current_bin
70     prev_visual = current_visual
71
72     current_visual = row["VISUAL_ID"]
73     current_bin = row["INTERFACE_BIN"]
74
75     if prev_visual != current_visual:
76         diferent_flag = 1
77     else:
78         diferent_flag = 0
79
80     # Nueva unidad
81     if diferent_flag == 1:
82         prev_bin = 0
83         total_units = total_units + 1
84         bin_switch_flag = 0
85
86         if(current_bin == 1):
87             # Unidades buenas, no las tomamos en cuenta
88             # Solid_index.append(index)
89             if(row['MODULE'] == 'HXV101'):
90                 SV1.append(current_visual)
91             elif(row['MODULE'] == 'HXV103'):
92                 SV3.append(current_visual)
93             elif(row['MODULE'] == 'HXV105'):
94                 SV5.append(current_visual)
95         else:
96             # Unidad de retest, hay que analizar
97             Current_retest_index.append(index)
98
99     # Unidad de retest
100    if diferent_flag == 0:
101        Current_retest_index.append(index)
102
103        # Es fallo s lido
104        if (prev_bin == current_bin) and (bin_switch_flag == 0) and
               (row['WITHIN_SESSION_LATEST_FLAG'] == 'Y'):
105            # Solid_index = Solid_index + Current_retest_index
```

```
                            # Sumamos a los s lidos
106             Current_retest_index.clear()
                                                    # Vaciamos buffer de
                   index
107             if(row['MODULE'] == 'HXV101'):
108                 SV1.append(current_visual)
109             elif(row['MODULE'] == 'HXV103'):
110                 SV3.append(current_visual)
111             elif(row['MODULE'] == 'HXV105'):
112                 SV5.append(current_visual)
113
114         # Es bin Switch
115         if (prev_bin != current_bin) or (bin_switch_flag == 1):
116             bin_switch_flag = 1
                                                        # Prendemos
                   bandera para que no se confunda con solidas
117             if (row["WITHIN_SESSION_LATEST_FLAG"] == 'Y') :
118                 Current_retest_index.clear()
                                                # Vaciamos el buffer de
                       index
119 Solid_visual = [SV1, SV3, SV5]
120 print('Se van a ignorar: ', int(len(Solid_visual[0])+int(len(
       Solid_visual[1]))+int(len(Solid_visual[2]))), ' l neas que son
       unidades s lidas.')
121 df_Switching = df
122 df_Switching.shape
123
124
125 # %% [markdown]
126 # ## Filtrado:
127 #
128 # - Conteo en una sola celda por un solo colateral
129 # - Posteriormente a las dem s celdas
130 # - Repote para los dem s tools
131
132 # %% [markdown]
133 # ##### Contamos "socketing" por colateral y promedio de test time
134
135 # %%
136
137 df_Switching_Backup = df_Switching.copy()
138
```

```
# Dataframe de salida del preprocesamiento de datos y entrada del algoritmo de ML
df_final = pd.DataFrame(columns=['Socketing', 'TIU', 'G/B_flag',
    'Test_Time', 'Bines_General', 'Bines_NLot',
    '1' , '2', '3', '4', '5', '6', '7', '8', '9', '10',
    '11', '12', '13', '14', '15', '16', '17', '18', '19', '20',
    '21', '22', '23', '24', '25', '26', '27', '28', '29', '30',
    '31', '32', '33', '34', '35', '36', '37', '38', '39', '40',
    '41', '42', '43', '44', '45', '46', '47', '48', '49', '50',
    '51', '52', '53', '54', '55', '56', '57', '58', '59', '60',
    '61', '62', '63', '64', '65', '66', '67', '68', '69', '70',
    '71', '72', '73', '74', '75', '76', '77', '78', '79', '80',
    '81', '82', '83', '84', '85', '86', '87', '88', '89', '90',
    '91', '92', '93', '94', '95', '96', '97', '98', '99'])

# Listas de maquinas y celdas donde se busca la informaci n de performance por colateral.
Tool_Number = ['HXV101', 'HXV103', 'HXV105']

Cell_Number = ['A101', 'A102', 'A201', 'A202', 'A301', 'A302', 'A401', 'A402', 'A501', 'A502',
            'B101', 'B102', 'B201', 'B202', 'B301', 'B302', 'B401', 'B402', 'B501', 'B502',
            'C101', 'C102', 'C201', 'C202', 'C301', 'C302', 'C401', 'C402', 'C501', 'C502']

n = 0       # Indice de fila a rellenar
prev_TIU = ''
current_TIU = ''
counter =0

# Recorremos el array para cada tool
for Tool in Tool_Number:

    print('Analizing tool ', Tool, '...')
    df_tool = df[df.MODULE.isin([Tool])]
    df_Switching_tool = df_Switching_Backup[df_Switching_Backup.MODULE.isin([Tool])]

    # Iteraci n por celda
    for Cell in Cell_Number:
        print('Analizing cell ', Cell, '...')
```

```
175         df_celda = df_tool[df_tool.SITE_ID.isin([Cell])]
176         df_Switching = df_Switching_tool[df_Switching_tool.SITE_ID.
               isin([Cell])]
177
178         # Primero lo har  para TIU, Aqu  cambiar amos a otros
               colaterales
179         df_TIU = df_celda.drop(['THERMAL_HEAD_ID'], axis=1)
180         df_TIU = df_TIU.drop(['DEVICE_TESTER_ID'], axis=1)
181         df_TIU = df_TIU.drop(['MODULE'], axis=1)
182         df_TIU = df_TIU.drop(['SITE_ID'], axis=1)
183
184         df_TIU = df_TIU.sort_index()
185         df_Switching = df_Switching.sort_index()
186         current_TIU_socketing = 1
187         test_time_prom = 0
188         DUT_index = []          #Aqui guardo los indices de la TIU
               que estamos analizando para luego sacar muestras buenas.
189         solid_search = 0        # To change search of solid visual
               between tools
190
191         ###################################
192                                         #
193         balance = int(sys.argv[3])      # Este valor quiere decir
               cuantos datos "Buenos" habr n por cada "Malo" EJ: 10/1
194         contador_balance = 0            # variable auxiliar de
               balance
195         relacion_balance = 0            # variable auxiliar de
               balance
196         relacion_balance_dinamico = 0   # variable auxiliar de
               balance
197                                         #
198         ###################################
199
200
201         # Recorremos el array y detectamos cambios de colateral (
               que suponemos como marginalidad)
202         for index, row in df_TIU.iterrows():
203             prev_LOT = ''
204             current_LOT = ''
205             Lot_History = [0]
206             prev_TIU = current_TIU
207             current_TIU = row['TESTER_INTERFACE_UNIT_ID']
```

```
208             current_Test_time= row['TEST_TIME']
209             current_date = row['DEVICE_END_DATE_TIME']
210
211             test_time_prom = test_time_prom + current_Test_time
212
213             # Seguimos en el mismo colateral
214             if prev_TIU == current_TIU:
215                 DUT_index.append(index)
216                 current_TIU_socketing = current_TIU_socketing + 1
217
218             # Diferente colateral, nueva fila
219             elif (prev_TIU != current_TIU) and (prev_TIU != ''):
220                 df_final.loc[str(n)] = np.zeros(105)
221                 df_final.loc[str(n), 'Socketing'] =
                       current_TIU_socketing
222                 df_final.loc[str(n), 'TIU'] = prev_TIU[-5:]
223                 df_final.loc[str(n), 'G/B_flag'] = 0
224                 df_final.loc[str(n), 'Test_Time'] = round(
                       test_time_prom / current_TIU_socketing, 3)
225
226                 # Buscamos bines (ignoramos s lidos)
227                 reg_index = []
228                 a_flag = 1
229
230                 relacion_balance = current_TIU_socketing/(balance +
                        2)  # Le sumamos 1 al balance para evitar el
                       caso donde ya est  malo el colateral
231                 contador_balance = 0            # variable auxiliar
                        de balance
232                 relacion_balance_dinamico = relacion_balance
233
234                 for index_2, row_2 in df_Switching.iterrows():
235                     counter += 1
236                     prev_LOT = current_LOT
237                     current_LOT = row_2['LOT']
238
239                     #####       Parte del c digo para calcular
                           filas de colaterales buenos      #####
240
241                     if((contador_balance == int(
                           relacion_balance_dinamico)) and (int(
                           relacion_balance_dinamico) <= int(
```

```
                relacion_balance*(balance))))):
242              df_final.loc[str(n), 'G/B_flag'] = 1
                                                  # Se
                    indica que es unidad buena
243              df_final.loc[str(n), 'Socketing'] = int(
                    relacion_balance_dinamico)
244              df_final.loc[str(n), 'Test_Time'] = row_2['
                    TEST_TIME']                  # usamos
                    test time actual (todos der an ser
                    similares)
245              df_final.loc[str(n), 'Bines_NLot'] = round(
                    sum(Lot_History)/len(Lot_History), 3)
                                # Metemos el promedio de
                    bines de fallo por lote
246              n += 1
247              df_final.loc[str(n)] = np.zeros(105)
                                                  #
                    creamos nueva fila
248              df_final.loc[str(n)] = df_final.loc[str(n
                    -1)]                          #
                    copiamos la anterior
249              df_final.loc[str(n), 'Socketing'] =
                    current_TIU_socketing          #
                    Rellenamos valores por defecto (el malo)
250              df_final.loc[str(n), 'G/B_flag'] = 0
251              df_final.loc[str(n), 'Test_Time'] = round(
                    test_time_prom / current_TIU_socketing,
                    3)
252              relacion_balance_dinamico =
                    relacion_balance_dinamico +
                    relacion_balance
253          else:
254              contador_balance = contador_balance + 1
255
256          #####

                 #####

257
258          if (row_2['TESTER_INTERFACE_UNIT_ID'] ==
                 prev_TIU) and (a_flag == 1):
259              reg_index.append(index_2)
260
```

```
261                         if(str(row_2['VISUAL_ID']) not in
                               Solid_visual[solid_search]):
262
263                             df_final.loc[str(n), str(int(row_2['
                                   INTERFACE_BIN']))] += 1          #
                                   sumamos a los bines
264                             df_final.loc[str(n), 'Bines_General']
                                   += 1                             #
                                   Sumamos al historial general de
                                   bines
265
266                             # Verificamos por lote (hacer promedio
                                   por lote)
267                             if(current_LOT == prev_LOT):
268                                 Lot_History[-1] += 1
                                                        # Sumamos a los
                                       bines de un mismo lote
269                             else:
270                                 Lot_History.append(1)
                                                       # Agregamos un
                                       lote nuevo
271                         else:
272                             Solid_visual[solid_search].remove(str(
                                   row_2['VISUAL_ID']))
                                                          # Eliminamos
                                   el visual que ya fue consultado
273                     else:
274                         a_flag = 0
275                 df_Switching_Backup = df_Switching_Backup.drop(
                       reg_index)
276                 df_Switching = df_Switching.drop(reg_index)
277
278                 df_final.loc[str(n), 'Bines_NLot'] = round(sum(
                       Lot_History)/len(Lot_History), 3)             #
                       Metemos el promedio de bines de fallo por lote
279
280                 test_time_prom = 0
281                 n = n+1
282                 current_TIU_socketing = 1
283                 reg_index = []
284
285         # Temporalmente terminamos con un cierto colateral que
```

```
        suponemos como bueno
286     df_final.loc[str(n)] = np.zeros(105)
287     df_final.loc[str(n),'Socketing'] = current_TIU_socketing
288     df_final.loc[str(n), 'TIU'] = current_TIU[-5:]
289     df_final.loc[str(n), 'G/B_flag'] = 1
290     df_final.loc[str(n), 'Test_Time'] = round(test_time_prom /
        current_TIU_socketing, 3)
291     a_flag = 1
292     prev_LOT = ''
293     current_LOT = ''
294     Lot_History = [0]
295     reg_index = []
296
297     # Buscamos bines
298     for index_2, row_2 in df_Switching.iterrows():
299         counter += 1
300         prev_LOT = current_LOT
301         current_LOT = row_2['LOT']
302
303         if (row_2['TESTER_INTERFACE_UNIT_ID'] == prev_TIU) and
            (a_flag == 1):
304             reg_index.append(index_2)
305
306             if(str(row_2['VISUAL_ID']) not in Solid_visual[
                solid_search]):
307
308                 df_final.loc[str(n), str(int(row_2['
                    INTERFACE_BIN']))] += 1              #
                    sumamos a los bines
309                 df_final.loc[str(n), 'Bines_General'] += 1
                                                  # Sumamos al
                    historial general de bines
310
311                 # Verificamos por lote (hacer promedio por lote
                    )
312                 if(current_LOT == prev_LOT):
313                     Lot_History[-1] += 1                #
                        Sumamos a los bines de un mismo lote
314                 else:
315                     Lot_History.append(1)               #
                        Agregamos un lote nuevo
316             else:
```

```
317                     Solid_visual[solid_search].remove(str(row_2['
                           VISUAL_ID'])) #
                           Eliminamos el visual que ya fue consultado
318             else:
319                 a_flag = 0
320         df_final.loc[str(n), 'Bines_NLot'] = round(sum(Lot_History)
               /len(Lot_History), 3)  # Metemos el promedio
               de bines de fallo por lote
321         test_time_prom = 0
322         df_Switching_Backup = df_Switching_Backup.drop(reg_index)
323         df_Switching = df_Switching.drop(reg_index)
324 
325         n = n+1
326         current_TIU_socketing = 1
327     # print(len(df_final))
328     solid_search = solid_search + 1
329     print("Done!")
330 
331 # print(counter)
332 total_analized = df_final['Socketing'].sum()
333 df_final
334 
335 
336 # %% [markdown]
337 # ### Filtramos condiciones especiales de los datos finales
338 
339 # %%
340 # Elimino todas las columnas que son cero (bines que nunca
       ocurrieron)
341 for o in range(99):
342     if(df_final[str(o+1)].sum() == 0):
343         del df_final[str(o+1)]
344 
345 # Elimino filas con colaterales nulos (no se corrieron unidades en
       la celda)
346 # Elimino filas con cero bines malos, estos indican que todos los
       malos que tuvieron fueron s lidos
347 # Eliminio filas "Buenas" que son los ultimos colaterales en
       m dulo (no se pueden clasificar aun)
348 # Elimino las filas con socketing cero o con socketing repetido (
       para las muestras de colaterales buenos)
349 empy_row = []
```

```
prev_socketing = 0
for index, row in df_final.iterrows():
    # print(row['Socketing'])
    # print(prev_socketing)
    if row['TIU'] == '':
        empy_row.append(index)
    elif row['Bines_General'] == 0:
        empy_row.append(index)
    elif (((row['G/B_flag'] == 1) and (row['Socketing'] ==
       prev_socketing)) | (row['Socketing'] == 0)):
        empy_row.append(index)
    prev_socketing = row['Socketing']
df_final = df_final.drop(empy_row)

df_final

# %% [markdown]
# Escribimos el archivo con los datos listos para alimentar el
    modelo

# %%
# Archivo en donde escribimos los resultados
# file_results_name = '\ML_input.xlsx'
excel_results_file = sys.argv[2]

# crear el objeto ExcelWriter
escrito = pd.ExcelWriter(excel_results_file)

# escribir el DataFrame en excel
print('Writing␣results␣file...')
df_final.to_excel(escrito, index=False)

# guardar el excel
escrito.save()
print('DONE')
```

Apéndice B

Código Master de Transformación de datos

```
# Archivo para aumatizar la lectura de bases de datos locales
import pandas as pd
import os
import time
# Medimos tiempo de ejecusi n
inicio = time.time()

DATA_DIR = "\Raw_Data"
RESULTS_DIR = "\Results"

Inputs_DB = ["\DB1.xlsx", "\DB2.xlsx", "\DB3.xlsx", "\DB4.xlsx", "\
    DB5.xlsx", "\DB6.xlsx", "\DB7.xlsx", "\DB8.xlsx", "\DB9.xlsx", "
    \DB10.xlsx", "\DB11.xlsx", "\DB12.xlsx", "\DB13.xlsx", "\DB14.
    xlsx"]
Output_DB = ["\T_out1.xlsx", "\T_out2.xlsx", "\T_out3.xlsx", "\
    T_out4.xlsx", "\T_out5.xlsx", "\T_out6.xlsx", "\T_out7.xlsx", "\
    T_out8.xlsx", "\T_out9.xlsx", "\T_out10.xlsx", "\T_out11.xlsx",
    "\T_out12.xlsx", "\T_out13.xlsx", "\T_out14.xlsx"]
Preprocessing_file = "C:/Users/felip/AppData/Local/Programs/Python/
    Python39/python.exe␣Data_Transformation.py"
current_dir = os.getcwd()
Rutabase = os.path.abspath(os.path.join(current_dir, os.pardir))
contador = 0
for inputDB in Inputs_DB:
    print("Analizing␣DataBase...")
    Rutarel_in = DATA_DIR + inputDB
    input = Rutabase + Rutarel_in
    Rutarel_out = RESULTS_DIR +  Output_DB[contador]
    contador += 1
```

```
23      output = Rutabase + Rutarel_out
24      balance = "5"
25      print('Input: ', input)
26      os.system(Preprocessing_file + ' ' + input + ' ' + output + ' '
            + balance)
27
28  fin = time.time()
29  print("Finished successfully!!")
30  print("Tiempo de ejecuci n: ", fin-inicio)
```

Apéndice C

Código Preprocesamiento de datos

```
# %% [markdown]
# ### Imports

# %%
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
from pickle import dump
from sklearn.manifold import TSNE

# %% [markdown]
# ### Data Integration

# %%

print("Merging␣results␣for␣data␣integration...")
df1 = pd.read_excel("../Results/T_out1.xlsx")
df2 = pd.read_excel("../Results/T_out2.xlsx")
df3 = pd.read_excel("../Results/T_out3.xlsx")
df4 = pd.read_excel("../Results/T_out4.xlsx")
df5 = pd.read_excel("../Results/T_out5.xlsx")
df6 = pd.read_excel("../Results/T_out6.xlsx")
df7 = pd.read_excel("../Results/T_out7.xlsx")
df8 = pd.read_excel("../Results/T_out8.xlsx")
```

```
df9 = pd.read_excel("../Results/T_out9.xlsx")
df10 = pd.read_excel("../Results/T_out10.xlsx")
df11 = pd.read_excel("../Results/T_out11.xlsx")
df12 = pd.read_excel("../Results/T_out12.xlsx")
df13 = pd.read_excel("../Results/T_out13.xlsx")
df14 = pd.read_excel("../Results/T_out14.xlsx")

data = pd.concat([df1, df2, df3, df4, df5, df6, df7, df8, df9, df10
    , df11, df12, df13, df14])

# data = pd.read_excel("../Results/ML_input.xlsx")
print('Done.')


# %% [markdown]
# ## Data Cleaning

# %%

print('Performing data cleaning...')
print('Inicial shape: ', data.shape)
print(data.columns)
# Rellenamos los espacios vac os con cero
data = data.fillna(0)

# Elimino la columna de nombre del TIU
data = data.drop(['TIU'], axis=1)

# Eliminamos la fila de bin 1 ya es un bin de resultado positivo
data = data.drop(['1'], axis=1)

# Elimino las columnas que tengan socketing mejor a 30
index = data[data["Socketing"]<30].index
data = data.drop(index)
index = 0

# Elimino las colunas que no tengan algun valor mayor a 5
for col in data.columns:
    good_flag = 0
    for n in data.loc[:,col]:
        if(int(n) >= 5):
            good_flag = 1
```

```
71      if((not good_flag) and (col != 'G/B_flag')):
72          data = data.drop([col], axis=1)
73
74  print('Final Shape: ', data.shape)
75
76
77  # %%
78  # Caracter sticas de los datos
79  data.describe().round(3)
80
81  # %% [markdown]
82  # ### Reducci n de dimensionalidad
83  # Se busca correlation de los datos para reducir las dimensiones,
        recordando que para naive bayes lo mejor es que las
        caracter sticas sean independientes entre si.
84
85  # %%
86  # calculate the correlations
87  correlations = data.corr()
88  correlations.to_excel("Correlation.xlsx")
89  # plot the heatmap
90  sns.heatmap(correlations, xticklabels=correlations.columns,
        yticklabels=correlations.columns, annot=True)
91
92  # plot the clustermap
93  sns.clustermap(correlations, xticklabels=correlations.columns,
        yticklabels=correlations.columns, annot=True)
94
95
96  # %% [markdown]
97  # Relaciones altas encontradas entre bines:
98  #
99  # - bin 51 y bin13: 0.75
100 # - Bin 44 y bin 14: 0.73
101 # - Bin 44 y Bin 20: 0.71
102 # - Bin54 y Bin48: 0.88
103 #
104 # Por experiencia del proceso sabemos que tanto el bin 13 como el
        51 est n asociados a fallos en este colateral por lo que decido
         dejar ambos como naive asumption
105 # se ve que el bin 44 representa bien la din mica del bin 14 y el
        20 por lo tanto se deja solo en 44
```

```
106 # y tambi n por proceso sabemos que el bin54 est  asociado a la
      TIU por lo tanto se eliminar  el bin 48.
107
108 # %%
109 data = data.drop(['20'], axis=1)
110 data = data.drop(['14'], axis=1)
111 data = data.drop(['48'], axis=1)
112
113 print('Nueva␣dimension:␣', data.shape)
114
115 # %%
116 data
117
118 # %% [markdown]
119 # ### Data Normalization
120
121 # %%
122 print('Begining␣with␣data␣normalization...')
123 Scaler = MinMaxScaler()
124 standard = StandardScaler()
125 buffer = data.copy()
126 columnas = data.columns
127 data = pd.DataFrame(Scaler.fit_transform(data), columns=columnas)
128 data_s = pd.DataFrame(standard.fit_transform(data), columns=
      columnas)
129
130 for i in range(len(data)):
131     data.iloc[i, 1] = int(buffer.iloc[i, 1])
132     data_s.iloc[i, 1] = int(buffer.iloc[i, 1])
133
134
135 # %%
136 # Guardamos scaler
137 dump(Scaler, open('scaler.pkl', 'wb'))
138
139 # %% [markdown]
140 # ### Visualizacion de datos
141
142 # %%
143 tsne = TSNE(n_components=2, random_state=0)
144 x_test_2d = tsne.fit_transform(data_s.drop(['G/B_flag'], axis=1))
145 labels = data_s['G/B_flag'].values
```

```
146 x_test_2d
147 plt.figure(figsize=(8,7))
148 plt.title("TIU␣marginal")
149 plt.scatter(x_test_2d[labels==0, 0], x_test_2d[labels == 0, 1], c='
      r', label='Bad', alpha=0.3)
150 plt.scatter(x_test_2d[labels==1, 0], x_test_2d[labels == 1, 1], c='
      b', label='Good', alpha=0.3)
151 plt.legend()
152 plt.show()
153
154 # %%
155 sns.pairplot(data.iloc[:,[0,1,2,4,5,9,10,23]], hue='G/B_flag')
156
157 # %% [markdown]
158 # Guardar resultados finales
159
160 # %%
161 # creating a new excel file and save the data
162 data.to_excel("../Results/ML_input.xlsx", index=False)
```

Apéndice D

Código Modelo Naive Bayes

```
# %% [markdown]
# # Naive Bayes Classifier

# %%
# import libraries
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os

from sklearn.naive_bayes import MultinomialNB
                                        # Para Modelo de Naive Bayers
    Multinomial
from sklearn.naive_bayes import GaussianNB
                                           # Para Modelo de Naive
    Bayers Multinomial
from sklearn.naive_bayes import BernoulliNB
                                          # Para Modelo de Naive
    Bayers Multinomial
from sklearn.metrics import recall_score, precision_score, f1_score
                  # Para medir rendimiento del modelo
from pickle import dump
from sklearn.model_selection import KFold

# %%
# importar datos
# Manejo de archivos
file_name = 'ML_input.xlsx'
```

```
current_dir = os.getcwd()
Rutabase = os.path.abspath(os.path.join(current_dir, os.pardir))
RESULTS_DIR = "\Results"
excel_filename = Rutabase + RESULTS_DIR +  '\ML_input.xlsx'
print('Analyzing document: ',excel_filename)
data = pd.read_excel(excel_filename)
data

# %%
# Separar datos entre entrenamiento y pruebas
tag = data['G/B_flag']
data_train = data.drop(['G/B_flag'], axis=1)
print(data_train.shape)
print(tag.shape)

# %%
# Creaci n de modelo
classifier_multinomial = MultinomialNB()
classifier_gaussian = GaussianNB()
classifier_bernuilli = BernoulliNB()

# Entrenamiento del modelo
classifier_multinomial.fit(data_train, tag)
classifier_gaussian.fit(data_train, tag)
classifier_bernuilli.fit(data_train, tag)

# save the model
dump(classifier_multinomial, open('model_MULT.pkl', 'wb'))
dump(classifier_bernuilli, open('model_BERNU.pkl', 'wb'))
dump(classifier_gaussian, open('model_GAUS.pkl', 'wb'))

# %% [markdown]
# ### Cross Validation

# %%
kf = KFold(n_splits = 10)
Bernuilli_score = []
Bernuilli_recall =[]
Bernuilli_precision = []
Bernuilli_f1 = []

Gaussian_score = []
```

```
65 Gaussian_recall =[]
66 Gaussian_precision = []
67 Gaussian_f1 = []
68
69 Multinomial_score = []
70 Multinomial_recall =[]
71 Multinomial_precision = []
72 Multinomial_f1 = []
73
74 for train_index, test_index in kf.split(data):
75     # Recalculando modelos
76     classifier_multinomial.fit(data_train.iloc[train_index], data.
       iloc[train_index,1])
77     classifier_bernuilli.fit(data_train.iloc[train_index], data.
       iloc[train_index,1])
78     classifier_multinomial.fit(data_train.iloc[train_index], data.
       iloc[train_index,1])
79
80     # Rendimiento multinomial
81     Multinomial_recall.append(recall_score(tag.iloc[test_index],
       classifier_multinomial.predict(data_train.iloc[test_index]))
       )
82     Multinomial_score.append(classifier_multinomial.score(
       data_train.iloc[test_index], tag.iloc[test_index]))
83     Multinomial_precision.append(precision_score(tag.iloc[
       test_index], classifier_multinomial.predict(data_train.iloc[
       test_index])))
84     Multinomial_f1.append(f1_score(tag.iloc[test_index],
       classifier_multinomial.predict(data_train.iloc[test_index]))
       )
85
86     # Rendimiento Gaussian
87     Gaussian_recall.append(recall_score(tag.iloc[test_index],
       classifier_gaussian.predict(data_train.iloc[test_index])))
88     Gaussian_score.append(classifier_gaussian.score(data_train.iloc
       [test_index], tag.iloc[test_index]))
89     Gaussian_precision.append(precision_score(tag.iloc[test_index],
        classifier_gaussian.predict(data_train.iloc[test_index])))
90     Gaussian_f1.append(f1_score(tag.iloc[test_index],
       classifier_gaussian.predict(data_train.iloc[test_index])))
91
92     # Rendimiento Bernuilli
```

```
93      Bernuilli_recall.append(recall_score(tag.iloc[test_index],
           classifier_bernuilli.predict(data_train.iloc[test_index])))
94      Bernuilli_score.append(classifier_bernuilli.score(data_train.
           iloc[test_index], tag.iloc[test_index]))
95      Bernuilli_precision.append(precision_score(tag.iloc[test_index
           ], classifier_bernuilli.predict(data_train.iloc[test_index])
           ))
96      Bernuilli_f1.append(f1_score(tag.iloc[test_index],
           classifier_bernuilli.predict(data_train.iloc[test_index])))
97
98  print('Multinomial␣score:␣', round(sum(Multinomial_score) / len(
       Multinomial_score),3)*100)
99  print('Multinomial␣recall:␣', round(sum(Multinomial_recall) / len(
       Multinomial_recall),3)*100)
100 print('Multinomial␣presicion:␣', round(sum(Multinomial_precision) /
        len(Multinomial_precision),3)*100)
101 print('Multinomial␣f1:␣', round(sum(Multinomial_f1) / len(
       Multinomial_f1),2)*100)
102
103 print('----------------------------------')
104
105 print('Gaussian␣score:␣', round(sum(Gaussian_score) / len(
       Gaussian_score),3)*100)
106 print('Gaussian␣recall:␣', round(sum(Gaussian_recall) / len(
       Gaussian_recall),3)*100)
107 print('Gaussian␣presicion:␣', round(sum(Gaussian_precision) / len(
       Gaussian_precision),3)*100)
108 print('Gaussian␣f1:␣', round(sum(Gaussian_f1) / len(Gaussian_f1),3)
       *100)
109
110 print('----------------------------------')
111
112 print('Bernuilli␣score:␣', round(sum(Bernuilli_score) / len(
       Bernuilli_score),3)*100)
113 print('Bernuilli␣recall:␣', round(sum(Bernuilli_recall) / len(
       Bernuilli_recall),3)*100)
114 print('Bernuilli␣presicion:␣', round(sum(Bernuilli_precision) / len
       (Bernuilli_precision),3)*100)
115 print('Bernuilli␣f1:␣', round(sum(Bernuilli_f1) / len(Bernuilli_f1)
       ,3)*100)
116
117 # %% [markdown]
```

```
118 # ### Estad sticas
119
120 # %%
121 ## Declaramos valores para el eje x
122 eje_x = ['Multinomial', 'Gaussian', 'Bernuilli']
123
124 eje_y = [round(sum(Multinomial_score) / len(Multinomial_score),3)
      *100, round(sum(Gaussian_score) / len(Gaussian_score),3)*100,
      round(sum(Bernuilli_score) / len(Bernuilli_score),3)*100]
125 plt.bar(eje_x, eje_y, color=['blue','red','green'])
126 plt.ylabel('Score')
127 plt.xlabel('Modelo')
128 plt.title('Naive␣Bayes')
129 plt.show()
130
131 # %%
132
133 eje_y = [ round(sum(Multinomial_recall) / len(Multinomial_recall)
      ,3)*100, round(sum(Gaussian_recall) / len(Gaussian_recall),3)
      *100, round(sum(Bernuilli_recall) / len(Bernuilli_recall),3)
      *100]
134 plt.bar(eje_x, eje_y, color=['blue','red','green'])
135 plt.ylabel('Recall')
136 plt.xlabel('Modelo')
137 plt.title('Naive␣Bayes')
138 plt.show()
139
140
141 # %%
142 eje_y = [ round(sum(Multinomial_precision) / len(
      Multinomial_precision),3)*100, round(sum(Gaussian_precision) /
      len(Gaussian_precision),3)*100, round(sum(Bernuilli_precision) /
       len(Bernuilli_precision),3)*100]
143 plt.bar(eje_x, eje_y, color=['blue','red','green'])
144 plt.ylabel('Precision')
145 plt.xlabel('Modelo')
146 plt.title('Naive␣Bayes')
147 plt.show()
148
149 # %%
150 eje_y = [ round(sum(Multinomial_f1) / len(Multinomial_f1),3)*100,
      round(sum(Gaussian_f1) / len(Gaussian_f1),3)*100, round(sum(
```

```
    Bernuilli_f1) / len(Bernuilli_f1),3)*100]
plt.bar(eje_x, eje_y, color=['blue','red','green'])
plt.ylabel('F1')
plt.xlabel('Modelo')
plt.title('Naive Bayes')
plt.show()
```

Apéndice E

Código de ejecusión para nuevos datos

```
# %% [markdown]
# ### Imports

# %%
# %matplotlib qt5
import pandas as pd
import numpy as np
import os
import sys
from pickle import load
import matplotlib.pyplot as plt

# %% [markdown]
# ### Preprocesamiento de los datos

# %%
print('Running preprocessing...')

# Manejo de archivos
file_name = "\Hands_on_tool.xlsx"
# file_name = str(sys.argv[1])  #Si queremos usarlo como argumento
DATA_DIR = "\Raw_Data"
RESULTS_DIR = "\Results"
current_dir = os.getcwd()
Rutabase = os.path.abspath(os.path.join(current_dir, os.pardir))
Rutarel_in = DATA_DIR + file_name
excel_filename = Rutabase + Rutarel_in
print('Analyzing document: ',excel_filename)

```

```
30 # Se importan los datos
31 data = pd.read_excel(excel_filename)
32 df = pd.DataFrame(data, columns= ['VISUAL_ID', '
       WITHIN_SESSION_SEQUENCE_NUMBER',
33                                   'WITHIN_SESSION_LATEST_FLAG', '
                                        INTERFACE_BIN',
34                                   'TESTER_INTERFACE_UNIT_ID', 'MODULE
                                        ', 'SITE_ID', 'TEST_TIME',
35                                   'DEVICE_END_DATE_TIME', 'LOT'])
36 # Ordenamos por fecha y hora
37 df = df.sort_values(by=['DEVICE_END_DATE_TIME'])
38 df = df.reset_index(drop=True)
39 df.shape
40
41 #### Filtrado de bines s lidos y conteo de bines por colateral
       general y por lote
42 # Recorremos el array para ver s lidos y los eliminaremos de la
       lista (lista nueva llamada df_Switching)
43 Solid_visual = []
44 SV1 = []
45 SV3 = []
46 SV5 = []
47
48 Current_retest_index = []
49 df = df.sort_values(by=['VISUAL_ID','WITHIN_SESSION_SEQUENCE_NUMBER
       ']) # Ordenamos por visual y n mero de secuencia
50 df_Switching = df.copy()
51 diferent_flag = 0
52 bin_switch_flag = 0
53 prev_visual = ''
54 prev_bin = 0
55 current_bin = 0
56 current_visual = ''
57 total_units = 0
58
59 print('Buscando␣unidades␣s lidas!')
60 # Recorremos los datos
61 for index, row in df.iterrows():
62     prev_bin = current_bin
63     prev_visual = current_visual
64
65     current_visual = row["VISUAL_ID"]
```

```
66      current_bin = row["INTERFACE_BIN"]
67
68      if prev_visual != current_visual:
69          diferent_flag = 1
70      else:
71          diferent_flag = 0
72
73      # Nueva unidad
74      if diferent_flag == 1:
75          prev_bin = 0
76          total_units = total_units + 1
77          bin_switch_flag = 0
78
79          if(current_bin == 1):
80              # Unidades buenas, no las tomamos en cuenta
81              if(row['MODULE'] == 'HXV101'):
82                  SV1.append(current_visual)
83              elif(row['MODULE'] == 'HXV103'):
84                  SV3.append(current_visual)
85              elif(row['MODULE'] == 'HXV105'):
86                  SV5.append(current_visual)
87          else:
88              # Unidad de retest, hay que analizar
89              Current_retest_index.append(index)
90
91      # Unidad de retest
92      if diferent_flag == 0:
93          Current_retest_index.append(index)
94
95          # Es fallo s lido
96          if (prev_bin == current_bin) and (bin_switch_flag == 0) and
                (row['WITHIN_SESSION_LATEST_FLAG'] == 'Y'):
97              Current_retest_index.clear()
                                                # Vaciamos buffer de
                    index
98              if(row['MODULE'] == 'HXV101'):
99                  SV1.append(current_visual)
100             elif(row['MODULE'] == 'HXV103'):
101                 SV3.append(current_visual)
102             elif(row['MODULE'] == 'HXV105'):
103                 SV5.append(current_visual)
104
```

```
105         # Es bin Switch
106         if (prev_bin != current_bin) or (bin_switch_flag == 1):
107             bin_switch_flag = 1
                                                    # Prendemos
                bandera para que no se confunda con solidas
108             if (row["WITHIN_SESSION_LATEST_FLAG"] == 'Y') :
109                 Current_retest_index.clear()
                                            # Vaciamos el buffer de
                    index
110 Solid_visual = [SV1, SV3, SV5]
111 print('Se van a ignorar: ', int(len(Solid_visual[0])+int(len(
    Solid_visual[1]))+int(len(Solid_visual[2]))), ' l neas que son
    unidades s lidas.')
112 df_Switching = df
113 df_Switching.shape
114
115 # Contamos "socketing" por colateral y promedio de test time
116 df_Switching_Backup = df_Switching.copy()
117 # Dataframe de salida del preprocesamiento de datos y entrada del
    algoritmo de ML
118 df_final = pd.DataFrame(columns=['Socketing', 'TIU', 'Tool', 'Cell'
    ,
119     'Test_Time', 'Bines_General', 'Bines_NLot',
120     '1' , '2', '3', '4', '5', '6', '7', '8', '9', '10',
121     '11', '12', '13', '14', '15', '16', '17', '18', '19', '20',
122     '21', '22', '23', '24', '25', '26', '27', '28', '29', '30',
123     '31', '32', '33', '34', '35', '36', '37', '38', '39', '40',
124     '41', '42', '43', '44', '45', '46', '47', '48', '49', '50',
125     '51', '52', '53', '54', '55', '56', '57', '58', '59', '60',
126     '61', '62', '63', '64', '65', '66', '67', '68', '69', '70',
127     '71', '72', '73', '74', '75', '76', '77', '78', '79', '80',
128     '81', '82', '83', '84', '85', '86', '87', '88', '89', '90',
129     '91', '92', '93', '94', '95', '96', '97', '98', '99'])
130
131 # Listas de maquinas y celdas donde se busca la informaci n de
    performance por colateral.
132 Tool_Number = ['HXV101', 'HXV103', 'HXV105']
133
134 Cell_Number = ['A101', 'A102', 'A201', 'A202', 'A301', 'A302', '
    A401', 'A402', 'A501', 'A502',
135             'B101', 'B102', 'B201', 'B202', 'B301', 'B302', 'B401',
                    'B402', 'B501', 'B502',
```

```
136             'C101', 'C102', 'C201', 'C202', 'C301', 'C302', 'C401',
                    'C402', 'C501', 'C502']
137
138 n = 0        # Indice de fila a rellenar
139 prev_TIU = ''
140 current_TIU = ''
141 counter =0
142
143 # Recorremos el array para cada tool
144 for Tool in Tool_Number:
145
146     print('Analizing␣tool␣', Tool, '...')
147     df_tool = df[df.MODULE.isin([Tool])]
148     df_Switching_tool = df_Switching_Backup[df_Switching_Backup.
          MODULE.isin([Tool])]
149
150     # Iteraci n por celda
151     for Cell in Cell_Number:
152         # print('Analizing cell ', Cell, '...')
153         df_celda = df_tool[df_tool.SITE_ID.isin([Cell])]
154         df_Switching = df_Switching_tool[df_Switching_tool.SITE_ID.
              isin([Cell])]
155
156         df_TIU = df_celda
157         df_TIU = df_TIU.sort_index()
158         df_Switching = df_Switching.sort_index()
159         current_TIU_socketing = 1
160         test_time_prom = 0
161         DUT_index = []          #Aqui guardo los indices de la TIU
              que estamos analizando para luego sacar muestras buenas.
162         solid_search = 0        # To change search of solid visual
              between tools
163
164         # Recorremos el array y detectamos cambios de colateral (
              que suponemos como marginalidad)
165         for index, row in df_TIU.iterrows():
166             prev_LOT = ''
167             current_LOT = ''
168             Lot_History = [0]
169             prev_TIU = current_TIU
170             current_TIU = row['TESTER_INTERFACE_UNIT_ID']
171             current_Test_time= row['TEST_TIME']
```

```
172             current_date = row['DEVICE_END_DATE_TIME']
173
174             test_time_prom = test_time_prom + current_Test_time
175
176             # Seguimos en el mismo colateral
177             if prev_TIU == current_TIU:
178                 DUT_index.append(index)
179                 current_TIU_socketing = current_TIU_socketing + 1
180
181             # Diferente colateral, nueva fila
182             elif (prev_TIU != current_TIU) and (prev_TIU != ''):
183                 test_time_prom = 0
184                 current_TIU_socketing = 1
185                 reg_index = []
186
187         # Temporalmente terminamos con un cierto colateral que
                suponemos como bueno
188         df_final.loc[str(n)] = np.zeros(106)
189         df_final.loc[str(n),'Socketing'] = current_TIU_socketing
190         df_final.loc[str(n), 'TIU'] = current_TIU[-5:]
191         df_final.loc[str(n), 'Test_Time'] = round(test_time_prom /
                current_TIU_socketing, 3)
192         df_final.loc[str(n), 'Tool'] = Tool
193         df_final.loc[str(n), 'Cell'] = Cell
194         a_flag = 1
195         prev_LOT = ''
196         current_LOT = ''
197         Lot_History = [0]
198         reg_index = []
199
200         # Buscamos bines
201         for index_2, row_2 in df_Switching.iterrows():
202             counter += 1
203             prev_LOT = current_LOT
204             current_LOT = row_2['LOT']
205
206             if (row_2['TESTER_INTERFACE_UNIT_ID'] == prev_TIU) and
                    (a_flag == 1):
207                 reg_index.append(index_2)
208
209                 if(str(row_2['VISUAL_ID']) not in Solid_visual[
                        solid_search]):
```

```
210
211                     df_final.loc[str(n), str(int(row_2['
                           INTERFACE_BIN']))] += 1              #
                           sumamos a los bines
212                     df_final.loc[str(n), 'Bines_General'] += 1
                                                  # Sumamos al
                           historial general de bines
213
214                     # Verificamos por lote (hacer promedio por lote
                           )
215                     if(current_LOT == prev_LOT):
216                         Lot_History[-1] += 1                 #
                               Sumamos a los bines de un mismo lote
217                     else:
218                         Lot_History.append(1)                #
                               Agregamos un lote nuevo
219                 else:
220                     Solid_visual[solid_search].remove(str(row_2['
                           VISUAL_ID'])) #
                           Eliminamos el visual que ya fue consultado
221             else:
222                 a_flag = 0
223         df_final.loc[str(n), 'Bines_NLot'] = round(sum(Lot_History)
               /len(Lot_History), 3)             # Metemos el promedio
               de bines de fallo por lote
224         test_time_prom = 0
225         df_Switching_Backup = df_Switching_Backup.drop(reg_index)
226         df_Switching = df_Switching.drop(reg_index)
227         n = n+1
228         current_TIU_socketing = 1
229         prev_TIU = ''
230         current_TIU = ''
231     solid_search = solid_search + 1
232     print("Done!")
233
234 total_analized = df_final['Socketing'].sum()
235 df_final
236
237
238 # %%
239 # Se les da forma a los datos para se analizados (necesario en
       modelos din micos)
```

```
# file_name = "\ML_input.xlsx"
# RESULTS_DIR = "\Results"
# current_dir = os.getcwd()
# Rutarel = RESULTS_DIR + file_name
# excel_filename = Rutabase + Rutarel
# # Se importan los datos
# data1 = pd.read_excel(excel_filename, nrows=1)
# data1 = data1.drop(['G/B_flag'],axis=1)
# correct_head = data1.columns

# Columnas de datos con los que se entren  al modelo de ML
correct_head = ['Socketing', 'Test_Time', 'Bines_General', 'Bines_NLot', '8', '9', '10',
       '11', '13', '15', '18', '19', '27', '28', '31', '35', '41', '42', '43',
       '44', '46', '47', '51', '54', '56', '60', '62', '64', '68', '92', '94',
       '97', '98', '99', '53']

evaluation_df = pd.DataFrame(columns=correct_head)
fila = []
for index, row in df_final.iterrows():
        for head in correct_head:
                fila.append(row[head])
        evaluation_df.loc[index] = fila
        fila = []
evaluation_df

# %%
# Se estandarizan los datos
Scaler = load(open('scaler.pkl', 'rb'))
evaluation_df = pd.DataFrame(Scaler.fit_transform(evaluation_df), columns=correct_head)
evaluation_df


# %% [markdown]
# ### Model

# %%
# Cargamos los modelos
classifier_multinomial = load(open('model_MULT.pkl', 'rb'))
```

```
278 classifier_gaussian = load(open('model_GAUS.pkl', 'rb'))
279 classifier_bernuilli = load(open('model_BERNU.pkl', 'rb'))
280
281 # Realizamos la predicci n
282 prediccion_gau = classifier_gaussian.predict(evaluation_df)
283 prediccion_multi = classifier_multinomial.predict(evaluation_df)
284 prediccion_bernu = classifier_bernuilli.predict(evaluation_df)
285 print('GAUSSIAN:')
286 print(prediccion_gau)
287 print('MULTINOMIAL:')
288 print(prediccion_multi)
289 print('BERNUILLI')
290 print(prediccion_bernu)
291
292 # %% [markdown]
293 # ### Visualizacion de resultados
294
295 # %%
296
297 ###### Modelo Elegido #######
298 ##############################
299 ChoosenOne = prediccion_multi
300 ##############################
301
302 # Para rearmar los indices en la visualizaci n
303 indexing1 = [0,1,10,11,20,21]    # A101, A102, B101, B102, C101,
      C102
304 indexing2 = [2,3,12,13,22,23]    # A201, A202, B201, B202, C201,
      C202
305 indexing3 = [4,5,14,15,24,25]    # A301, A302, B301, B302, C301,
      C302
306 indexing4 = [6,7,16,17,26,27]    # A401, A402, B401, B402, C401,
      C402
307 indexing5 = [8,9,18,19,28,29]    # A501, A502, B501, B502, C501,
      C502
308 t101 = []
309 t103 = []
310 t105 = []
311 counter = 0
312
313 # Segregamos y acomodamos los datos segun m quina
314 for i in range(30):
```

```
    t101.append(ChoosenOne[i])
    t103.append(ChoosenOne[i+30])
    t105.append(ChoosenOne[i+60])
Matriz_101 =
   [[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]

Matriz_103 =
   [[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]

Matriz_105 =
   [[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]

counter = 0
for i in indexing5:
    Matriz_101[0][counter]=t101[i]
    Matriz_103[0][counter]=t103[i]
    Matriz_105[0][counter]=t105[i]
    counter +=1
counter = 0
for i in indexing4:
    Matriz_101[1][counter]=t101[i]
    Matriz_103[1][counter]=t103[i]
    Matriz_105[1][counter]=t105[i]
    counter +=1
counter = 0
for i in indexing3:
    Matriz_101[2][counter]=t101[i]
    Matriz_103[2][counter]=t103[i]
    Matriz_105[2][counter]=t105[i]
    counter +=1
counter = 0
for i in indexing2:
    Matriz_101[3][counter]=t101[i]
    Matriz_103[3][counter]=t103[i]
    Matriz_105[3][counter]=t105[i]
    counter +=1
counter = 0
for i in indexing1:
    Matriz_101[4][counter]=t101[i]
    Matriz_103[4][counter]=t103[i]
    Matriz_105[4][counter]=t105[i]
    counter +=1
```

```
351
352 print('HXV101:␣', Matriz_101)
353 print('HXV103:␣', Matriz_103)
354 print('HXV105:␣', Matriz_105)
355
356
357 # %% [markdown]
358 # #### Gr f i c a s
359
360 # %%
361 fig = plt.figure(figsize=(25,15))
362 fig.suptitle('\nTester␣Interface␣Unit␣Marginality\n', fontsize=50,
      y=0.8, color='b')
363 matriz_1 = pd.DataFrame(Matriz_101)
364 plt.subplot(131)
365 plt.matshow(matriz_1, cmap='Greys',vmin=0, fignum=False, vmax=1)
366 plt.title('HXV101', fontsize=20, color='c')
367 plt.text(-0.2, 0, 'A501');plt.text(0.8, 0, 'A502');plt.text(1.8, 0,
       'B501');plt.text(2.8, 0, 'B502');plt.text(3.8, 0, 'C501');plt.
      text(4.8, 0, 'C502')
368 plt.text(-0.2, 1, 'A401');plt.text(0.8, 1, 'A402');plt.text(1.8, 1,
       'B401');plt.text(2.8, 1, 'B402');plt.text(3.8, 1, 'C401');plt.
      text(4.8, 1, 'C402')
369 plt.text(-0.2, 2, 'A301');plt.text(0.8, 2, 'A302');plt.text(1.8, 2,
       'B301');plt.text(2.8, 2, 'B302');plt.text(3.8, 2, 'C301');plt.
      text(4.8, 2, 'C302')
370 plt.text(-0.2, 3, 'A201');plt.text(0.8, 3, 'A202');plt.text(1.8, 3,
       'B201');plt.text(2.8, 3, 'B202');plt.text(3.8, 3, 'C201');plt.
      text(4.8, 3, 'C202')
371 plt.text(-0.2, 4, 'A101');plt.text(0.8, 4, 'A102');plt.text(1.8, 4,
       'B101');plt.text(2.8, 4, 'B102');plt.text(3.8, 4, 'C101');plt.
      text(4.8, 4, 'C102')
372 plt.gca().set_xticks([x - 0.5 for x in plt.gca().get_xticks()][1:],
       minor='true')
373 plt.gca().set_yticks([y - 0.5 for y in plt.gca().get_yticks()][1:],
       minor='true')
374 plt.xticks([])
375 plt.yticks([])
376 plt.grid(which='minor')
377
378 matriz_3 = pd.DataFrame(Matriz_103)
379 plt.subplot(132)
```

```
plt.matshow(matriz_3, cmap='Greys', vmin=0, fignum=False, vmax=1)
plt.title('HXV103', fontsize=20, color='c')
plt.text(-0.2, 0, 'A501');plt.text(0.8, 0, 'A502');plt.text(1.8, 0,
    'B501');plt.text(2.8, 0, 'B502');plt.text(3.8, 0, 'C501');plt.
    text(4.8, 0, 'C502')
plt.text(-0.2, 1, 'A401');plt.text(0.8, 1, 'A402');plt.text(1.8, 1,
    'B401');plt.text(2.8, 1, 'B402');plt.text(3.8, 1, 'C401');plt.
    text(4.8, 1, 'C402')
plt.text(-0.2, 2, 'A301');plt.text(0.8, 2, 'A302');plt.text(1.8, 2,
    'B301');plt.text(2.8, 2, 'B302');plt.text(3.8, 2, 'C301');plt.
    text(4.8, 2, 'C302')
plt.text(-0.2, 3, 'A201');plt.text(0.8, 3, 'A202');plt.text(1.8, 3,
    'B201');plt.text(2.8, 3, 'B202');plt.text(3.8, 3, 'C201');plt.
    text(4.8, 3, 'C202')
plt.text(-0.2, 4, 'A101');plt.text(0.8, 4, 'A102');plt.text(1.8, 4,
    'B101');plt.text(2.8, 4, 'B102');plt.text(3.8, 4, 'C101');plt.
    text(4.8, 4, 'C102')
plt.gca().set_xticks([x - 0.5 for x in plt.gca().get_xticks()][1:],
    minor='true')
plt.gca().set_yticks([y - 0.5 for y in plt.gca().get_yticks()][1:],
    minor='true')
plt.xticks([])
plt.yticks([])
plt.grid(which='minor')
plt.subplot(133)
matriz_5 = pd.DataFrame(Matriz_105)
plt.matshow(matriz_5, cmap='Greys', vmin=0, fignum=False, vmax=1)
plt.title('HXV105', fontsize=20, color='c')
plt.text(-0.2, 0, 'A501');plt.text(0.8, 0, 'A502');plt.text(1.8, 0,
    'B501');plt.text(2.8, 0, 'B502');plt.text(3.8, 0, 'C501');plt.
    text(4.8, 0, 'C502')
plt.text(-0.2, 1, 'A401');plt.text(0.8, 1, 'A402');plt.text(1.8, 1,
    'B401');plt.text(2.8, 1, 'B402');plt.text(3.8, 1, 'C401');plt.
    text(4.8, 1, 'C402')
plt.text(-0.2, 2, 'A301');plt.text(0.8, 2, 'A302');plt.text(1.8, 2,
    'B301');plt.text(2.8, 2, 'B302');plt.text(3.8, 2, 'C301');plt.
    text(4.8, 2, 'C302')
plt.text(-0.2, 3, 'A201');plt.text(0.8, 3, 'A202');plt.text(1.8, 3,
    'B201');plt.text(2.8, 3, 'B202');plt.text(3.8, 3, 'C201');plt.
    text(4.8, 3, 'C202')
plt.text(-0.2, 4, 'A101');plt.text(0.8, 4, 'A102');plt.text(1.8, 4,
    'B101');plt.text(2.8, 4, 'B102');plt.text(3.8, 4, 'C101');plt.
```

```
    text(4.8, 4, 'C102')
plt.gca().set_xticks([x - 0.5 for x in plt.gca().get_xticks()][1:],
    minor='true')
plt.gca().set_yticks([y - 0.5 for y in plt.gca().get_yticks()][1:],
    minor='true')
plt.xticks([])
plt.yticks([])
plt.grid(which='minor')
plt.show()
```

Bibliografía

[1] Matheus Pinto Arratia. *Identificación de impactos en el fuselaje de un avión utilizando algoritmos de aprendizaje de máquinas.* Universidad de chile, 2019.

[2] Daniel Berrar. *Bayes' Theorem and Naive Bayes Classifier.* Tokyo institude of Technology, Tokyo, Japón, 2019.

[3] Angel Choez Franco. *Análisis de las características de los tipos de algoritmos de clustering en el aprendizaje no supervizado.* Universidad técnica de Babahoyo, 2022.

[4] A. Ravi G. Rebala and S. Churiwala. *An Introduction to Machine Learning.* Springer Nature Switzerland, San Jose, CA, USA, 2019.

[5] Alexander Huertas Mora. *Algoritmos de aprendizaje supervisado utilizando datos de monitoreo de condiciones: Un estudio para el pron ´ostico de fallas en máquinas.* Universidad Santo Tomás, Bogotá, Colombia, 2020.

[6] Gonzalo Martínez Muñoz. *Clasificación mediante conjuntos.* Universidad Autónoma de Madrid, 2006.

[7] J. Luengo S. García, S. Ramírez and F. Herrera. *Big Data: Preprocesamiento y calidad de datos.* Universidad de Granada, España, 2016.

[8] Geoffrey I. Webb. *Naive Bayes.* Monash University, Melbourne, Victoria, 2011.