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

Índice general

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 pero 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 de 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 ciertos 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 mayor 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 machine learnig de tiempo 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 por cada colateral.
- Evaluar el funcionamiento dentro de un conjunto de colaterales según su celda para toda una misma máquina.
- Crear una interfaz de usuario que muestre los datos necesarios y que permita una fácil comprensión y rápida para el usuario.
- 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 testing 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.

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. [2]

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. [3] Algunos de los algoritmos de aprendizaje automático supervisado son:

- clasificación
- Regresión

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. [3] Algunos de los algoritmos de aprendizaje no supervisado son:

- segmentación (clustering)
- Reducción de dimensionalidad

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. [3] De este existen también subconjuntos de deep learning como los siguientes:

- Perceptrón multicapa (MLP)
- Red neuronales convolucionales (CNN)
- 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 B$_{Z}$ un evento "B"dado que sucede .^{A}", dicha relación se expresa de la siguiente manera. [1]

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

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. [5]

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 Retestz 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. [4]

Apéndice A

Código Preprocesamiento de datos

```
# Librerías a utilizar

import pandas as pd
import numpy as np
import os

# Manejo de archivos
file_name = 'Project_First_Data.xlsx'
current_file = os.path.abspath(os.path.dirname(file_name))
excel_filename = current_file + '\Project_First_Data.xlsx'

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',
                                  'WITHIN_SESSION_LATEST_FLAG', 'INTERFACE_BIN',
                                  'TESTER_INTERFACE_UNIT_ID', 'THERMAL_HEAD_ID',
                                  'DEVICE_TESTER_ID', 'MODULE', 'SITE_ID', 'TEST_TIME',
                                  'DEVICE_END_DATE_TIME', 'LOT'])
# Ordenamos por fecha y hora
df = df.sort_values(by=['DEVICE_END_DATE_TIME'])
df = df.reset_index(drop=True)

Solid_index = []
Current_retest_index = []

df = df.sort_values(by=['VISUAL_ID','WITHIN_SESSION_SEQUENCE_NUMBER'])
df_Switching = df.copy()
```

```
diferent_flag = 0
bin_switch_flag = 0

prev_visual = ''
prev_bin = 0
current_bin = 0
current_visual = ''

total_units = 0

# Recorremos los datos
for index, row in df.iterrows():
    prev_bin = current_bin
    prev_visual = current_visual

    current_visual = row["VISUAL_ID"]
    current_bin = row["INTERFACE_BIN"]

    if prev_visual != current_visual:
        diferent_flag = 1
    else:
        diferent_flag = 0

    # Nueva unidad
    if diferent_flag == 1:
        prev_bin = 0
        total_units = total_units + 1
        bin_switch_flag = 0

        if(current_bin == 1):
            # Unidades buenas, no las tomamos en cuenta
           Solid_index.append(index)
        else:
            # Unidad de retest, no la tomamos encuenta
            Current_retest_index.append(index)

    # Unidad de retest
    if diferent_flag == 0:
        Current_retest_index.append(index)

        # Es fallo sólido
```

```
        if(prev_bin == current_bin)and(bin_switch_flag == 0)and(row['WITHIN_SESSION_LATEST_
            Solid_index = Solid_index + Current_retest_index
            Current_retest_index.clear()
        # Es bin Switch
        if (prev_bin != current_bin) or (bin_switch_flag == 1):
            bin_switch_flag = 1
            if (row["WITHIN_SESSION_LATEST_FLAG"] == 'Y') :
                Current_retest_index.clear()

print('Se van a eliminar: ', len(Solid_index), ' líneas que son unidades sólidas.')
df_Switching = df.drop(Solid_index)
df_Switching_Backup = df_Switching.copy()

# 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 información de performance por colateral.
Tool_Number = ['HXV101', 'HXV103', 'HXV105']
Cell_Number = ['A101', 'A102', 'A201', 'A202', 'A301', 'A302', 'A401', 'A402', 'A501', 'A50
            '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 = ''
# 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])]
    # Iteración por celda
    for Cell in Cell_Number:
        df_celda = df_tool[df_tool.SITE_ID.isin([Cell])]
```

```
df_Switching = df_Switching_tool[df_Switching_tool.SITE_ID.isin([Cell])]

# Primero lo haré para TIU, Aquí cambiaríamos a otros colaterales
df_TIU = df_celda.drop(['THERMAL_HEAD_ID'], axis=1)
df_TIU = df_TIU.drop(['DEVICE_TESTER_ID'], axis=1)
df_TIU = df_TIU.drop(['MODULE'], axis=1)
df_TIU = df_TIU.drop(['SITE_ID'], axis=1)
df_TIU = df_TIU.sort_index()
df_Switching = df_Switching.sort_index()
current_TIU_socketing = 1
test_time_prom = 0

# Recorremos el array y detectamos cambios de colateral
for index, row in df_TIU.iterrows():
    prev_LOT = ''
    current_LOT = ''
    Lot_History = [0]
    prev_TIU = current_TIU
    current_TIU = row['TESTER_INTERFACE_UNIT_ID']
    current_Test_time= row['TEST_TIME']
    current_date = row['DEVICE_END_DATE_TIME']

    test_time_prom = test_time_prom + current_Test_time
    # Seguimos en el mismo colateral
    if prev_TIU == current_TIU:
        current_TIU_socketing = current_TIU_socketing + 1

    # Diferente colateral, nueva fila
    elif (prev_TIU != current_TIU) and (prev_TIU != ''):
        df_final.loc[str(n)] = np.zeros(105)
        df_final.loc[str(n), 'Socketing'] = current_TIU_socketing
        df_final.loc[str(n), 'TIU'] = prev_TIU[-5:]
        df_final.loc[str(n), 'G/B_flag'] = 'B'
        df_final.loc[str(n), 'Test_Time']=round(test_time_prom/current_TIU_socketing,3)
        # Buscamos bines
        reg_index = []
        a_flag = 1

        for index_2, row_2 in df_Switching.iterrows():
            prev_LOT = current_LOT
            current_LOT = row_2['LOT']
            if (row_2['TESTER_INTERFACE_UNIT_ID'] == prev_TIU) and (a_flag == 1):
```

```
                    # print(index_2)
                    # print(index)
                    df_final.loc[str(n), str(row_2['INTERFACE_BIN'])] += 1            #
                    df_final.loc[str(n), 'Bines_General'] += 1
                    reg_index.append(index_2)

                    # Verificamos por lote (hacer promedio por lote)
                    if(current_LOT == prev_LOT):
                        Lot_History[-1] += 1
                    else:
                        Lot_History.append(1)
                else:
                    a_flag = 0
            df_Switching_Backup = df_Switching_Backup.drop(reg_index)
            df_Switching = df_Switching.drop(reg_index)
            df_final.loc[str(n),'Bines_NLot']=round(sum(Lot_History)/len(Lot_History),3
            test_time_prom = 0
            n = n+1
            current_TIU_socketing = 1
            reg_index = []

    # Temporalmente terminamos con un cierto colateral que suponemos como bueno
    df_final.loc[str(n)] = np.zeros(105)
    df_final.loc[str(n),'Socketing'] = current_TIU_socketing
    df_final.loc[str(n), 'TIU'] = current_TIU[-5:]
    df_final.loc[str(n), 'G/B_flag'] = 'G'
    df_final.loc[str(n), 'Test_Time'] = round(test_time_prom / current_TIU_socketing, 3
    a_flag = 1
    prev_LOT = ''
    current_LOT = ''
    Lot_History = [0]
    reg_index = []
    # Buscamos bines
    for index_2, row_2 in df_Switching.iterrows():
        prev_LOT = current_LOT
        current_LOT = row_2['LOT']

        if (row_2['TESTER_INTERFACE_UNIT_ID'] == prev_TIU) and (a_flag == 1):
            df_final.loc[str(n), str(row_2['INTERFACE_BIN'])] += 1
            df_final.loc[str(n), 'Bines_General'] += 1
            reg_index.append(index_2)
```

```
                # Verificamos por lote (hacer promedio por lote)
                if(current_LOT == prev_LOT):
                    Lot_History[-1] += 1
                else:
                    Lot_History.append(1)
            else:
                a_flag = 0
        df_final.loc[str(n), 'Bines_NLot'] = round(sum(Lot_History)/len(Lot_History), 3)
        test_time_prom = 0
        df_Switching_Backup = df_Switching_Backup.drop(reg_index)
        df_Switching = df_Switching.drop(reg_index)

        n = n+1
        current_TIU_socketing = 1
    print(len(df_final))
total_analized = df_final['Socketing'].sum()

empy_row = []
for index, row in df_final.iterrows():
    if row['TIU'] == '':
        empy_row.append(index)
    elif row['Bines_General'] == 0:
        empy_row.append(index)
    elif row['G/B_flag'] == 'G':
        empy_row.append(index)
df_final = df_final.drop(empy_row)

# Archivo en donde escribimos los resultados
file_results_name = '\ML_input.xlsx'
excel_results_file = current_file + file_results_name
# crear el objeto ExcelWriter
escrito = pd.ExcelWriter(excel_results_file)
# escribir el DataFrame en excel
df_final.to_excel(escrito)
# guardar el excel
escrito.save()
```

Bibliografía

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

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

[3] 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.

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

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