UNIVERSIDAD DE ANTIOQUIA

1 8 0 3

Estudio integral de equipos en subestaciones eléctricas ISA intercolombia

Luis José Molina Truyot

Universidad de Antioquia
Facultad de Ciencias Exactas y Naturales
Instituto de Matemáticas
Medellín, Colombia
2023

Estudio integral de equipos en subestaciones eléctricas ISA intercolombia

Luis José Molina Truyot

Trabajo de grado presentado como requisito parcial para optar al título de:
Matemático

Danilo Bedoya Valencia
Orientador Interno, Instituto de Matemáticas

Guillermo Atanasio Fonseca Pacheco
Orientador externo, ISA intercolombia

Universidad de Antioquia
Facultad de Ciencias Exactas y Naturales
Instituto de Matemáticas
Medellín, Colombia
2023

Resumen

En el contexto actual, ISA Intercolombia está utilizando algoritmos de lógica difusa y booleana en la plataforma SAP un software de gestión empresarial multiplataforma, para calificar sus equipos. Sin embargo, estos algoritmos han experimentado fallas difíciles de solucionar en la mayoría de los casos debido a la naturaleza compleja del entorno de programación además de ser algoritmos poco flexibles en ciertas necesidades de mantenimiento. Con el objetivo de abordar estos problemas, se plantea llevar a cabo un estudio exhaustivo de los algoritmos y las reglas de decisión correspondientes a cada equipo, con el fin de desarrollar una infraestructura capaz de calificar y generar notificaciones prioritarias de mantenimiento para estos activos de la empresa. Se comenzó con los CTS (transformadores de corriente), lo que proporciona un punto de partida para el análisis de los demás equipos puesto que a pesar de que estos presentan diferencias marcadas en su funcionamiento, comparten reglas de diagnóstico que siguen una lógica similar. Luego, se ha extraído la información necesaria de la base de datos de SAP y después de un proceso de depuración fueron usados para el entrenamiento de distintos modelos de clasificación, como árboles de decisión, bosques aleatorios y Gradient Boosting, con el propósito de determinar la técnica de aprendizaje automático más apropiada para abordar este problema complejo debido a la gran cantidad de reglas involucradas. Finalmente, se ha elegido Gradient Boosting como el modelo más adecuado para el problema. Se han comparado varios algoritmos de Gradient Boosting desarrollados por diferentes grupos, y se han realizado pruebas necesarias para corroborar que el modelo es capaz de calificar los equipos de manera precisa sin obstaculizar las labores de mantenimiento de ISA.

Palabras clave: Machine learning, Transformadores,ISA,Mantenimiento,Reglas de decisión,Arbol de decisión, Gradient Boosting, Depuración de datos, Estadistica, Boosting.

Abstract

In the current context, ISA Intercolombia is using fuzzy logic and boolean algorithms in the SAP platform, a cross-platform enterprise management software, to assess its equipment. However, these algorithms have experienced difficult-to-solve failures in most cases due to the complex nature of the programming environment, in addition to being inflexible algorithms for certain maintenance needs. In order to address these problems, a comprehensive study of the algorithms and decision rules corresponding to each piece of equipment is proposed, in order to develop an infrastructure capable of rating and generating prioritized maintenance notifications for these company assets. We started with the CTS (current transformers), which provides a starting point for analyzing the other equipment, since although they have marked differences in their operation, they share diagnostic rules that follow a similar logic. Next, the necessary information was extracted from the SAP database and, after a debugging process, it was used to train different classification models such as decision trees, random forests, and Gradient Boosting, in order to determine the most appropriate machine learning technique to address this complex problem due to the large number of involved rules. Finally, Gradient Boosting has been chosen as the most suitable model for the problem. Several Gradient Boosting algorithms developed by different groups were compared, and necessary tests were performed to confirm that the model is capable of accurately rating the equipment without hindering ISA's maintenance tasks. keywords. **Keywords: Machine learning, Transformad,ISA,Manteinance,Decision rules,Decision trees, Gradient Boosting, Data Wrangling, Statistics, Boosting**

Estudio integral de equipos en subestaciones eléctricas ISA intercolombia

Luis José Molina Truyot *

30 de junio de 2023

*E-mail: luis.molina@udea.edu.co, Instituto de Matemáticas, Universidad de Antioquia, Medellín, Colombia.

Contenido

Resumen **3**

1. Introducción **6**

2. Marco Teórico **7**
2.1. Aprendizaje Automatico 7
2.1.1. Terminologia del Aprendizaje automatico . . . 7
2.2. Arboles de decisión 8
2.2.1. Arboles de clasificación 9
2.2.2. Ventajas y desventajas de los arboles de decisión 10
2.3. Métodos de ensemble 11
2.3.1. Bagging 11
2.3.2. Boosting 13
2.3.3. Algotimo para Forward Stagewise Additive Modeling 15
2.3.4. Boosting para arboles de clasificación 16
2.4. Gradient Boost 17
2.4.1. Algoritmo para Gradient Tree Boost 18
2.5. Importancia de las variables predictoras 20
2.6. Metodo de la transformada inversa 21
2.6.1. Algoritmo método transformada inversa 21
2.7. Importancia por permutación 21
2.7.1. Incremento de la pureza de nodos 22
2.8. Grid Search 22
2.9. Metricas 22
2.10. Curvas ROC 23

3. Metodología **25**
3.1. Identificación del problema 25
3.2. Data Wrangling 28
3.2.1. Manejo de datos faltantes 35
3.3. Analisis de datos 37
3.3.1. Variables Categóricas 37
3.3.2. Variables Continuas 40
3.4. Estudio general de modelos de clasificación con los datos de SAP 44
3.4.1. Arboles de decisión 44
3.4.2. Bosques aleatorios 49
3.4.3. Clasificador XGBoost 51

4. Creación de datos sinteticos **55**

5. Resultados finales **57**
5.1. XGBOOST 57
5.2. LigthGBM 62
5.3. Capacidad predictiva 65

6. Conclusión **66**

1. Introducción

En el panorama energético de Colombia, ISA Intercolombia juega un papel crucial como encargada del transporte y distribución de la energía en todo el país. Esta tarea es posible gracias a una robusta infraestructura compuesta por subestaciones, torres de energía y otros componentes esenciales que facilitan los procesos intermedios necesarios para asegurar el suministro adecuado de energía en los hogares de la nación. Sin embargo, estos procesos intermedios, entre otros, dependen de una amplia variedad de equipos que requieren un monitoreo constante para garantizar su correcto funcionamiento. Actualmente, el mantenimiento de estos equipos se basa en avisos generados por la plataforma SAP, la cual utiliza algoritmos individuales para evaluar cada tipo de equipo.

No obstante, estos algoritmos, desarrollados hace más de 10 años, están presentando problemas que afectan la integridad del personal y la calidad del servicio energético. En conjunto con estas dificultades, los algoritmos carecen de flexibilidad y presentan lentitud, además de arrojar en ocasiones resultados incorrectos que obstaculizan las tareas de mantenimiento. Ante estos problemas operativos y aprovechando la transición tecnológica en curso, surge la propuesta de estudiar estos algoritmos y replicarlos en la sintaxis de Python o evaluar la viabilidad de desarrollar modelos de aprendizaje automático para crear calificadores con reglas más complejas, que brinden una mayor flexibilidad en la resolución de problemas y agilicen las labores de mantenimiento.

El objetivo principal es desarrollar modelos capaces de aprender estas reglas y aprovechar el potencial de los modelos de aprendizaje automático para mejorar la flexibilidad y precisión de las calificaciones, lo cual es esencial debido al funcionamiento diverso de los equipos. Estos modelos también permitirán optimizar las labores de mantenimiento al obtener resultados rápidos, y su capacidad de reentrenamiento facilitará la incorporación de nuevas reglas sin importar su complejidad.

Como punto de partida, examinaremos los algoritmos de calificación utilizados en los transformadores de corriente sin capacitador, los cuales presentan algoritmos de calificación considerablemente complejos y, según la consideración de los expertos, limitados en cuanto a sus capacidades para satisfacer las necesidades específicas de estos equipos. Para ello, se ha optado por emplear el lenguaje de programación Python como nueva plataforma para el desarrollo de cualquier algoritmo.

Ahora, analizaremos las variables de mayor relevancia para la calificación, con el objetivo de comprender la naturaleza del problema y las relaciones entre dichas variables. Luego, profundizaremos en el estudio de las reglas de calificación de estos equipos y en las necesidades actuales, como la calificación basada en registros históricos o la capacidad de distinguir entre grupos de equipos A y B, con el propósito de comprender la jerarquía de las reglas y los posibles escenarios de calificación en el entorno real. Tras diversas pruebas, se llega a la conclusión de que los modelos basados en Gradient Boosting son los más robustos para este tipo de escenario.

Sin embargo, este hallazgo no marcó el fin de la investigación, ya que ningún modelo era capaz de clasificar adecuadamente la mayoría de los casos, lo cual se debía a la insuficiencia de datos para que el modelo aprendiera todas las reglas. Conscientes de esta situación, y contando ya con un modelo sólido y capaz de aprender estas reglas, se decidió generar datos sintéticos con el propósito de cubrir todos los posibles casos de calificación, aprovechando la capacidad de los modelos equipados con Gradient Boosting para aprender de manera efectiva.

Una vez consolidados todos los modelos y algoritmos necesarios para la calificación de los diferentes tipos de equipos de ISA Intercolombia, se implementarán en una infraestructura integrada en SAP, lo que automatizará los procesos de procesamiento de datos, calificación y priorización de avisos.

2. Marco Teórico

2.1. Aprendizaje Automatico

El Machine Learning, o Aprendizaje Automático, es una rama de la inteligencia artificial que se enfoca en el desarrollo de algoritmos y modelos computacionales que permiten a las máquinas aprender y mejorar automáticamente a través de la experiencia, sin necesidad de ser programadas explícitamente para cada tarea específica. En lugar de seguir instrucciones rígidas, los sistemas de Machine Learning utilizan datos y ejemplos para aprender patrones y tomar decisiones o hacer predicciones. Existen varias ramas o subcampos dentro del campo del Machine Learning, cada uno con sus propias técnicas, enfoques y aplicaciones específicas. Algunas de las principales ramas del Machine Learning son las siguientes:

- **Aprendizaje Supervisado (Supervised Learning):** En esta rama, los modelos de Machine Learning se entrenan utilizando datos de entrada junto con las correspondientes etiquetas o respuestas deseadas. El objetivo es aprender una función que pueda mapear correctamente nuevas instancias de entrada a las salidas deseadas. Ejemplos comunes de algoritmos de aprendizaje supervisado incluyen las Máquinas de Vectores de Soporte (SVM), los Árboles de Decisión y las Redes Neuronales.
- **Aprendizaje No Supervisado (Unsupervised Learning):** Aquí, los modelos de Machine Learning se entrenan en datos sin etiquetas o respuestas conocidas. El objetivo principal es descubrir patrones ocultos, estructuras subyacentes o agrupamientos en los datos. Ejemplos de algoritmos de aprendizaje no supervisado incluyen el clustering (agrupamiento), la reducción de dimensionalidad y las redes generativas.
- **Aprendizaje por Refuerzo (Reinforcement Learning):** Esta rama se centra en desarrollar algoritmos que aprenden a través de la interacción con un entorno y la retroalimentación en forma de recompensas o castigos. El agente de aprendizaje toma decisiones secuenciales con el objetivo de maximizar las recompensas a largo plazo. El Aprendizaje por Refuerzo se aplica en problemas como los juegos de estrategia, la robótica y la optimización de recursos.
- **Aprendizaje Profundo (Deep Learning):** Es un subcampo del Machine Learning que se enfoca en el uso de redes neuronales artificiales con múltiples capas (redes neuronales profundas). Estas redes son capaces de aprender representaciones jerárquicas de los datos y han demostrado un rendimiento sobresaliente en áreas como el reconocimiento de imágenes, el procesamiento de lenguaje natural y el procesamiento de voz.

2.1.1. Terminologia del Aprendizaje automatico

En el aprendizaje automatico, los actores principales son las variables predictoras $X_1, ..., X_n$ y la variable respuesta Y. Las variables respuestas varian su naturaleza segun el tipo de problema que se esté trabajando. Un tipo de variable respuesta son las cuantitativas, las cuales puden ser valores discretos ($\{0, 1, 2, .., ..\}$) o continuos (valores reales) y ademas pueden ser cualitativas tal es como el genero, caracteristicas fisicas, estado de salud de algun equipo (bueno o malo) entre otros casos. Esta distinción en el tipo de variables respuesta ha dado lugar a una convención para denominar las tareas de predicción: regresión cuando predecimos resultados cuantitativos, y clasificación cuando predecimos resultados cualitativos. Analogamente, las variables predictoras tambien según el problema pueden tener estas dos naturalezas, dando lugar a distinciones en los tipos de métodos que se utilizan para la predicciónm. Algunos métodos se definen de forma más natural para las variables de entrada cuantitativas, otros de forma más natural para las cualitativas y otros para ambas.

Los modelos de aprendizaje automatico supervisado necesitan que que las variables sean numericas. Con las variables cuantitativas no hay ningún problema, pero para las variables cualitativas se necesita realizar un paso extra antes de continuar con el proceso de estudio de los datos. Por lo general, estas variables se codifican con numeros enteros, por ejemplo,cuando sólo hay dos clases o categorías, como "éxito.º "fracaso", "sobrevivió.º "murió". A menudo se representan mediante un único dígito o bit binario como 0 o 1, o bien mediante -1 y 1. Por razones que se harán evidentes, estos códigos numéricos se denominan a veces objetivos. Cuando hay más de dos categorías, existen varias alternativas. La más útil y utilizada es la codificación mediante variables dummy que toman los valores 0 o 1 para indicar la ausencia o presencia de algún efecto categórico que pueda esperarse que cambie el resultado, y se represetan en forma de vector. En caso de que existan 4 clases, serán vectores $4x1$ en el que la i-esima entrada indica posee 1 e la i-esima clase a la que pertence la entrada y el resto de entradas es 0 ([0,1,0,0] indica que es la clase 2) o simplemente codificar cada clase con algun numero entero, por ejemplo, clase 1 como 0, clase 2 como 1, clase 3 como 2.

Para construir reglas de predicción necesitamos datos, y a menudo muchos. Por lo tanto, se dispone de un conjunto de medidas (x_i, y_i) $i = 1, ..., N$, conocido como datos de entrenamiento y con estos datos se constuiran las reglas para las predicción

2.2. Arboles de decisión

Los métodos estadísticos y de machine learning basados en árboles engloban a un conjunto de técnicas supervisadas no paramétricas que consiguen estratificar o segmentar el espacio predictor en una serie de regiones simples. Para hacer una predicción de una observación determinada, se suele utilizar la media o la moda de las observaciones de entrenamiento de la región a la que pertenece. Dado que el conjunto de reglas de división utilizadas para segmentar el espacio predictor puede resumirse en un árbol, este tipo de enfoques se conocen como métodos de árbol de decisión.

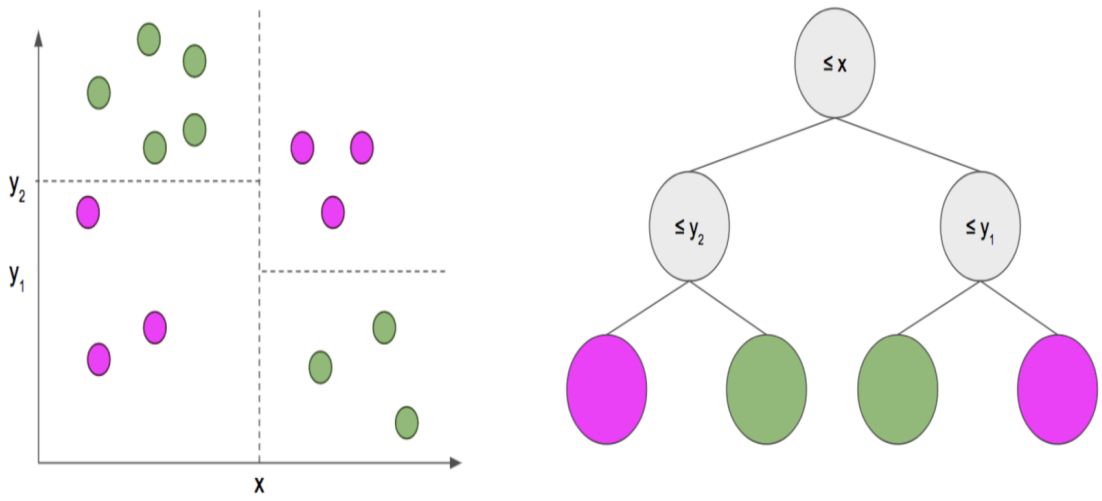


Figura 1: Arbol de decisión con sus regiones.

El proceso de construcción de un arbol de decisión, se puede resumir en dos pasos y se puede observar en la figura 1:

- Se divide el espacio predictor, es decir, el conjunto de todos los posibles valores de las variables $X_1, ..., X_p$ en J regiones distintas que no se solapan $R_1, ..., R_J$
- Para toda observación que cae en la región R_j, se realiza la misma predicción, el cual es simplemente la media de los valores respuesta para las observaciones de entrenamiento en R_j.

Existen dos tipos de arboles los cuales son los arboles de regresión y los arboles de clasificación. El exito de la identificación del tipo de arbol adecuado para el problema depende de la naturelza de la variable respuesta y dependiendo del tipo de arbol, la construcción de las regiones R_j puede variar.

2.2.1. Arboles de clasificación

Los arboles de clasificacion consisten en modelos que dado un conjunto de variables, estos son capaces de predecir una respuesta cualitativa. Para un árbol de clasificación, predecimos que cada observación pertenece a la clase de observaciones de entrenamiento más frecuente en la región a la que pertenece. Al interpretar los resultados de un árbol de clasificación, a menudo estamos interesados no solo en la predicción de la clase correspondiente a una región terminal en particular, sino también en las proporciones de clase entre las observaciones de entrenamiento que caen en esa región.

Consideremos p variables de entrada y una variable de respuesta para cada una de las N observaciones: es decir, $(x_i; y_i)$ para $i = 1, 2, .., N$ con $x_i = (x_{i1}, x_{i2}, .., x_{ip})$. El algoritmo necesita decidir automáticamente las variables de división y los puntos de división, así como la topología (forma) que debe tener el árbol. Supongamos primero que se tiene una partición en M regiones $R_1, R_2, ..., R_M$, y se modela la respuesta como una constante c_m en cada región.

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m) \tag{1}$$

Ahora se requiere en el algoritmo del árbol los criterios para dividir los nodos y podar el árbol. En un nodo m, que representa una región R_m con N_m observaciones, sea

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k) \tag{2}$$

donde $N_m = |\{x_i \in R_m\}|$ la proporción de observaciones de entrenamiento en la m-ésima región que pertenecen a la k-ésima clase. Se clasifican las observaciones en el nodo m en la clase $k(m) = \underset{k}{\text{argmax}} \hat{p}_{mk}$, que es la clase mayoritaria en el nodo m. Sin embargo, resulta que el error de clasificación no es suficientemente sensible para el crecimiento del árbol, y en la práctica tenemos otras dos medidas diferentes $Q_m(T)$ de la impureza del nodo que incluyen lo siguiente:

- **Misclassification error**

$$\frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)} \tag{3}$$

- **Indice Gini**

$$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{4}$$

- **Cross-entropy or deviance**

$$\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk} \tag{5}$$

Cuando se construye un árbol de clasificación, se suele utilizar el índice de Gini o la entropía cruzada para evaluar la calidad de una división concreta, ya que estos dos enfoques son más sensibles a la pureza de los nodos que la tasa de error de clasificación. Para el podado de los arboles se puede usar cualquiera de estas medidas, pero el mas usado para este proposito es la tasa de error de clasificación.

Es evidente que un árbol muy grande podría ajustarse demasiado a los datos, mientras que un árbol pequeño podría no captar la estructura importante.

La estrategia preferida es hacer crecer un árbol grande T_0, deteniendo el proceso de división sólo cuando se alcanza un tamaño mínimo de nodo (digamos 5). A continuación, este árbol grande se poda utilizando la poda de complejidad de costes que describimos a continuación.

Definimos un subárbol $T \subset T_0$ como cualquier árbol que se puede obtener por podando T_0, es decir, colapsando cualquier número de sus nodos internos (no terminales). (no terminales). Indexamos los nodos terminales por m, con el nodo m representando la región R_m. Sea $|T|$ el número de nodos terminales en T. Sea

$$N_m = |\{x_i \in R_m\}|$$

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$

y un Q_m elegido. Definimos el criteio cost complexity

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T| \tag{6}$$

La idea es encontrar, para cada α, el subárbol $T \subset T_0$ que minimice $C_\alpha(T)$.

El parámetro de ajuste α rige el equilibrio entre el tamaño del árbol y su ajuste a los datos. Los valores grandes de α dan como resultado árboles más pequeños T_α, y a la inversa para valores más pequeños de α. Como sugiere la notación, con $\alpha = 0$ la solución es el árbol completo T_0.

2.2.2. Ventajas y desventajas de los arboles de decisión

- Facil interpretabilidad y un funcionamiento cercano al proceso de la toma de decisión de los seres humanos.
- Los arboles de decisión pueden ser graficados y son muy sencillos de interpretar, especialmente si son arboles pequeños.
- Los arboles pueden manejar facilmente predictores cualittivos sin la necesidad de variables dummy, por lo que codifcar suele ser suficiente.
- Son sensibles a datos de entrenamiento desbalanceados (una de las clases domina sobre las demás), por lo que forzamente toca simular el resto de clases.
- No son capaces de extrapolar fuera del rango de los predictores observado en los datos de entrenamiento.
- La capacidad predictiva de los modelos basados en un único árbol es bastante inferior a la conseguida con otros modelos. Esto es debido a su tendencia al overfitting y alta varianza. Sin embargo, existen técnicas más complejas(bagging, random forest, boosting), consiguen mejorar este problema.

- No se ven muy influenciados por outliers.

2.3. Métodos de ensemble

Cuando se incrementa la complejidad de un modelo, se le otorga una mayor capacidad para adaptarse a las observaciones, lo que resulta en una reducción del sesgo y en una mejora de su capacidad predictiva. No obstante, una vez que se alcanza un nivel específico de flexibilidad, surge el problema de sobreajuste (overfitting), en el cual el modelo se ajusta tanto a los datos de entrenamiento que es incapaz de predecir de manera precisa nuevas observaciones. El modelo óptimo es aquel que logra un balance adecuado entre el sesgo y la varianza. Usualmente, los árboles de decisión pequeños con pocas ramificaciones presentan una varianza baja, pero al mismo tiempo, no logran capturar adecuadamente la compleja relación entre las variables, lo que se traduce en un alto sesgo. En contraste, los árboles de decisión grandes se ajustan intensamente a los datos de entrenamiento, lo que resulta en un sesgo reducido, pero también en una varianza considerable. Una manera de abordar esta problemática son los métodos de ensemble.

La idea de los métodos de ensemble es construir un modelo de predicción combinando los puntos fuertes de una colección de modelos base más sencillos, con el objetivo de lograr un equilibro entre bias y varianza, consiguiendo así mejores predicciones que cualquiera de los modelos individuales originales. Los métodos de ensemble pueden dividirse en dos tareas: desarrollar una población de aprendices base a partir de los datos de entrenamiento y combinarlos para formar el predictor compuesto.

Dos de los tipos de ensemble más utilizados son el bagging y el boosting.

2.3.1. Bagging

Los árboles de decisión discutidos anteriormente sufren de alta varianza. Esto significa que si dividimos los datos de entrenamiento en dos partes al azar y ajustamos un árbol de decisión en ambas mitades, los resultados obtenidos podrían ser bastante diferentes. En contraste, un procedimiento con baja varianza producirá resultados similares si se aplica repetidamente a conjuntos de datos distintos. La regresión lineal tiende a tener baja varianza cuando la relación entre el número de observaciones n y el número de variables predictoras p es moderadamente grande. Un procedimiento de uso general para reducir la varianza de un método de aprendizaje estadístico es el boostraping agregation o bagging. El bagging está basado en el boostraping, el cual consiste en una técnica estadística que se utiliza para estimar la incertidumbre o la variabilidad de una muestra de datos basandose en el concepto de muestreo con reemplazo, donde se generan múltiples muestras de tamaño igual al de la muestra original a partir de esta misma muestra. El bagging es especialmente útil y se utiliza con frecuencia en el contexto de los árboles de decisión.

Consideremos un conjunto de n observaciones independientes $\{Z_1,...,Z_n\}$, cada una con varianza σ^2, la varianza de la media $\bar{Z}$ de las observaciones está dada por $\frac{\sigma^2}{n}$. En otras palabras, el promedio de un conjunto de observaciones reduce la varianza. Por lo tanto, una forma natural de reducir la varianza y, en consecuencia, aumentar la precisión de predicción de un método de aprendizaje estadístico es tomar muchos conjuntos de entrenamiento de la población, construir un modelo de predicción por separado utilizando cada conjunto de entrenamiento y promediar las predicciones resultantes. En otras palabras, podríamos calcular $\{\hat{f}_1(x),...,\hat{f}_B(x)\}$ utilizando B conjuntos de entrenamiento separados y promediarlos para obtener un único modelo de aprendizaje estadístico de baja varianza, dado por

$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^b(x) \tag{7}$$

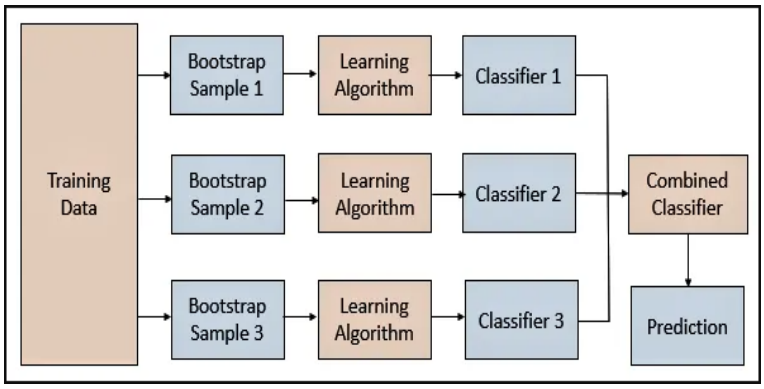


Figura 2: Bagging

Esto no es práctico porque generalmente no tenemos acceso a múltiples conjuntos de entrenamiento. En su lugar, podemos utilizar el bootstrap, tomando muestras repetidas del conjunto de datos de entrenamiento (único). En este enfoque, generamos B conjuntos de datos de entrenamiento bootstrap diferentes. Luego entrenamos nuestro método en el b-ésimo conjunto de datos de entrenamiento bootstrap para obtener $\hat{f}^{*b}(x)$ y finalmente promediamos todas las predicciones para obtener

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x) \tag{8}$$

Si bien el bagging puede mejorar las predicciones en muchos métodos de regresión, es especialmente útil para los árboles de decisión. Para aplicar bagging a los árboles de clasificación, simplemente construimos B árboles de clasificación utilizando B conjuntos de entrenamiento bootstrap. Luego, podemos registrar la clase predicha por cada uno de los B árboles y realizar una votación mayoritaria: la predicción general será la clase más común entre las B predicciones.

Un ejemplo muy famoso de bagging son los bosques aleatorios o Random forest. Los bosques aleatorios proporcionan una mejora sobre los árboles agregados mediante una pequeña modificación aleatoria que descorrelaciona los árboles. Al igual que en la técnica de Bagging, construimos un conjunto de árboles de decisión utilizando muestras de entrenamiento bootstrap. Pero al construir estos árboles de decisión, cada vez que se considera una división en un árbol, se elige una muestra aleatoria de m predictores como candidatos a la división del conjunto completo de p predictores.

La división se permite utilizar solo uno de esos m predictores. Se toma una muestra nueva de m predictores en cada división, y típicamente elegimos $m \approx \sqrt{p}$, es decir, el número de predictores considerados en cada división es aproximadamente igual a la raíz cuadrada del número total de predictores. En otras palabras, al construir un bosque aleatorio, en cada división del árbol, el algoritmo ni siquiera puede considerar la mayoría de los predictores disponibles. Supongamos que hay un predictor muy fuerte en el conjunto de datos, junto con otros predictores moderadamente fuertes. Entonces, en la colección de árboles agrupados, la mayoría o todos los árboles utilizarán este predictor fuerte en la primera división. En consecuencia, todos los árboles agrupados se verán muy similares entre sí. Por lo tanto, las predicciones de los árboles estarán altamente correlacionadas. Desafortunadamente, promediar muchas cantidades altamente correlacionadas no conduce a una reducción tan grande de la varianza como promediar muchas cantidades no correlacionadas. En particular, esto significa que la técnica de bagging no llevará a una reducción sustancial de la varianza en comparación con un solo árbol en esta configuración.

Los bosques aleatorios superan este problema al forzar que cada división considere solo un subcon-

junto de los predictores. Por lo tanto, en promedio, $\frac{p-m}{p}$ de las divisiones ni siquiera considerarán el predictor fuerte, lo que permite que otros predictores tengan más oportunidades. Podemos pensar en este proceso como una decorrelación de los árboles, lo que hace que el promedio de los árboles resultantes sea menos variable y, por lo tanto, más confiable. La diferencia principal entre la bagging y los bosques aleatorios es la elección del tamaño del subconjunto de predictores m. Por ejemplo, si se construye un bosque aleatorio utilizando $m = p$, entonces esto equivale simplemente a la agregación. Utilizar un valor pequeño de m al construir un bosque aleatorio suele ser beneficioso cuando tenemos un gran número de predictores correlacionados

2.3.2. Boosting

El bagging consiste en crear múltiples copias del conjunto de datos de entrenamiento original utilizando el bootstrap, ajustar un árbol de decisión distinto a cada copia y, a continuación, combinar todos los árboles para crear un único modelo predictivo. En particular, cada árbol se construye a partir de un conjunto de datos bootstrap, independiente de los demás árboles. El Boosting funciona de forma similar, con la diferencia de que los árboles crecen secuencialmente como se observa en la figura 3: cada árbol crece utilizando la información de los árboles anteriores. El Boosting no implica un muestreo bootstrap, sino que cada árbol se ajusta a una versión modificada del conjunto de datos original.

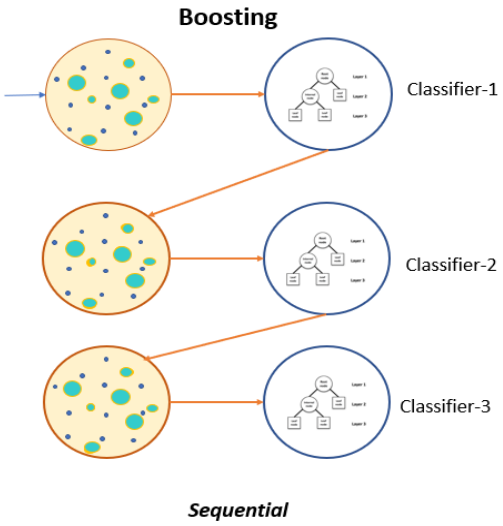


Figura 3: boosting

Para empezar a describir esta tecnica de aprendizaje, se hablará de AdaBoost (Adaptative Boosting), que fue desarrollado por parte de Yoav Freund y Robert Schapire. Considere un problema de clasificación binaria donde la variable respuesta Y está codificada de tal manera que $Y \in \{-1, 1\}$. Dado un vector de variables predictoras X, un clasificador G(X) produce una predicción que toma uno de los dos valores posibles. La tasa de error en la muestra de entrenamiento es:

$$\bar{\texttt{err}} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq G(x_i)) \tag{9}$$

donde I es la función indicatriz y la tasa de error esperada en las predicciones futuras es $E_{XY}I(Y \neq G(X)$.

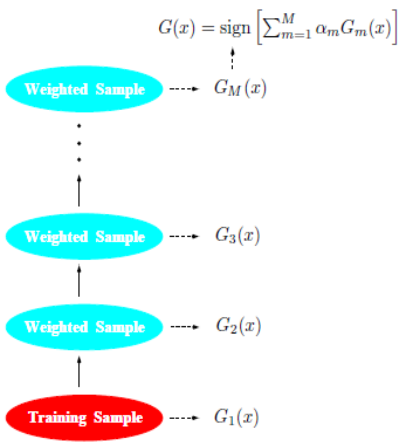


Figura 4: AdaBoost

Un clasificador débil es aquel cuya tasa de error es sólo ligeramente superior a la de la adivinación aleatoria. El propósito del boosting es aplicar secuencialmente el algoritmo de clasificación débil a versiones de los datos modificadas repetidamente, produciendo así una sucesión de clasificadores debiles $G_m(x), m = 1,2,...,M$. En la figura 4 se muestra un esquema del algoritmo anteriormente descrito.

Las predicciones de todos estos clasificadores debiles son combinados mediante un voto mayoritario ponderado para obtener la predicción final:

$$G(x) = \texttt{sign}(\sum_{m=1}^{M} \alpha_m G_m(x)) \tag{10}$$

Aqui $\alpha_1,..,\alpha_M$ son calculados por el algoritmo de Boosting y ponderan la contribución de cada $G_m(x)$ respectivo. Su efecto es dar mayor influencia a los clasificadores más precisos de la secuencia.

Las modificaciones de los datos en cada paso de boosting consisten en aplicar pesos $w_1, w_2, ..., w_N$ a cada una de las observaciones de entrenamiento (x_i, y_i), donde $i = 1,2,...,N$. Inicialmente, todos los pesos se establecen en $w_i = \frac{1}{N}$, de modo que el primer paso simplemente entrena el clasificador con los datos de la manera habitual. Para cada iteración sucesiva $m = 2,3,...,M$, los pesos de las observaciones se modifican individualmente y se vuelve a aplicar el algoritmo de clasificación a las observaciones ponderadas.

En el paso m, las observaciones que fueron clasificadas incorrectamente por el clasificador $G_{m-1}(x)$ inducido en el paso anterior tienen sus pesos incrementados, mientras que los pesos se reducen para aquellas que fueron clasificadas correctamente. De esta manera, a medida que avanzan las iteraciones, las observaciones que son difíciles de clasificar correctamente reciben cada vez más atención en la secuencia. Cada clasificador sucesivo se ve así obligado a concentrarse en aquellas observaciones de entrenamiento que no fueron correctamente clasificadas por los clasificadores anteriores en la secuencia.

Algoritmo AdaBoost

1. Inicializar los pesos de observación $w_i = \frac{1}{N}, i = 1,2,..,N$.
2. Para $m = 1$ a M:
    a. Ajustar un clasificador G_m a los datos de entrenamiento usando los pesos w_i

b. Calcula

$$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$

c. Calcula $\alpha_m = log(\frac{1-err_m}{err_m})$

d. Establece $w_i \cdot exp[\alpha_m \cdot I(y_i \neq G_m(x_i)], i = 1,2,..,N$

3. La salida será $G(x) = sign(\sum_{m=1}^{M} \alpha_m G_m(x))$

El boosting es una manera de ajustar una expansión additiva (10) en un conjunto de funciones "base.[el]ementales.Aquí las funciones de base son los clasificadores individuales $G_m(x) en \{-1,1\}$. De forma más general, las expansiones de las funciones base poseen la forma

$$f(x) = \sum_{m=1}^{M} \beta_m b(x;\gamma_m), \tag{11}$$

Donde β_m, $m = 1,2,..,M$ son los coeficientes de la expansión, y $b(x;\gamma) \in R$ suelen ser funciones simples de multivariadas x, caracterizadas por un conjunto de parámetros γ.

Normalmente, estos modelos se ajustan minimizando una función de pérdida promediada sobre los datos de entrenamiento, como el error cuadrático o una función de pérdida basada en la verosimilitud,

$$\min_{\beta_m,\gamma_m} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \beta_m b(x_i;\gamma_m)). \tag{12}$$

2.3.3. Algotimo para Forward Stagewise Additive Modeling

- Initialize $f_0(x) = 0$.
- Para $m = 1$ a M

  a Calcula

  $$(\beta_m, \gamma_m) = \underset{\beta,\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma))$$

  b Set $f_m(x) = f_{m-1}(x) + \beta_m b(x;\gamma_m$

Forward stagewise modeling aproxima la solución a (12) añadiendo secuencialmente nuevas funciones de base a la expansión sin ajustar los parámetros y coeficientes de las ya añadidas. Este algoritmo es descrito en la sección 2.3.3. En cada interacción m, se resuelve la función base óptima $b(x;\gamma_m)$ y el coeficiente correspondiente β_m para añadir a la expansión actual $f_{m-1}(x)$. Esto produce $f_m(x)$, y el proceso se repite. Los términos añadidos anteriormente no se modifican.

Para el error cuadratico

$$L(y, f(x)) = (y - f(x))^2$$

se tiene

$$L(y_i, f_{m-1}(x_i) + \beta b(x_i,\gamma)) = (y_i - f_{m-1}(x_i) - \beta b(x_i,\gamma))^2$$

$$(r_{im} - \beta b(x_i, \gamma))^2$$

donde $r_{im} = y_i - f_{m-1}(x_i)$

es simplemente el residuo del modelo actual en la i-ésima observación. Así, para el error cuadrático, el término $\beta_m b(x; \gamma_m)$ que mejor se ajusta a los residuos actuales se añade a la expansión en cada paso. Esta idea es la base de la potenciación de la regresión por "mínimos cuadrados". Sin embargo, el error cuadrático no suele ser una buena opción para la clasificación; de ahí la necesidad de considerar otros criterios de pérdida.

2.3.4. Boosting para arboles de clasificación

Los arboles de clasificación dividen el espacio de todos los valores conjuntos de las variables predictoras en regiones disjuntas $R_j, j = 1, 2, .., J$, representadas por los nodos terminales del árbol. Se asigna una constante γ_j a cada una de estas regiones y la regla de predicción es

$$x \in R_j \rightarrow f(x) = \gamma j$$

Asi, los arboles pueden ser expresados como

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j) \tag{13}$$

Con parametros $\Theta = \{R_j, \gamma_k\}_1^J$. J se trata usualmente como un "meta-parametro". Estos parametros son encontrados minimizando el riesgo empirico

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j) \tag{14}$$

Este problema de optimización se divide en dos partes

- **Encontrar γ_j dado R_j:** Dado el conjunto de regiones Rj, para estimar y_j a menudo se utiliza la estimación puntual $\hat{\gamma}_j = \hat{y_j}$, que es simplemente la media de los valores de y que se encuentran en la región R_j. Para una función de pérdida de clasificación incorrecta, $\hat{\gamma}_j$ es la clase modal de las observaciones que caen en la región R_j.
- **Encontrar R_j:** Encontrar los R_j implica también estimar los γ_j. Una estrategia típica consiste en utilizar un algoritmo de particionamiento recursivo para encontrar los R_j. Además, a veces es necesario aproximar $\hat{\Theta}$ (14) mediante un criterio más conveniente para optimizar los R_j:

$$\tilde{\Theta} = \underset{\Theta}{\operatorname{argmin}} \sum_{j=1}^{N} \tilde{L}(y_i, T(x_i; \Theta)) \tag{15}$$

Luego, dado $\hat{R}j = \tilde{R}j$, los γ_j pueden estimarse con mayor precisión utilizando el criterio original.

Para los arboles de clasificación, el indice Gini reemplaza la perdida por misclasification en el crecimiento de los arboles. Ahora, el modelo reforzado(boosted) es la suma de todos los m arboles y está dado por:

$$f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m) \tag{16}$$

la cual se deduce dado a "Forward Stagewise Additive Modeling" el cual es una tecnica para minimizar funciones de perdida basadas en la verosimilitud. En cada paso del "Forward Stagewise" hay que resolver

$$\hat{\Theta}_m = \underset{\Theta_m}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i;\Theta_m)) \tag{17}$$

(14.29)

para el conjunto de regiones y constantes $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$ del siguiente árbol, dado el modelo actual $f_{m-1}(x)$.

Dadas las regiones R_{jm},encontrar las constantes óptimas γ_{jm} en cada región se logra solucionando

$$\hat{\gamma}_{jm} = \underset{\gamma_{jm}}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm}) \tag{18}$$

2.4. Gradient Boost

Un modelo de árboles Gradient Boosting se construye por etapas como en otros métodos de refuerzo, pero generaliza los otros métodos al permitir la optimización de una función de pérdida diferenciable arbitraria. La construcción del gradient booting se realizó a partir de tecnicas de optimización numerica. En analogía con la optimización numérica, se pueden derivarse algoritmos aproximados para resolver (17) con cualquier criterio de pérdida diferenciable. La pérdida en el uso de f(x) para predecir y en los datos de entrenamiento es

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)) \tag{19}$$

El objetivo es minimizar $L(f)$ con respecto a f, donde aquí $f(x)$ está restringido a ser una suma de árboles (16). Ignorando esta restricción, minimizar (19) puede verse como una optimización numérica

$$\hat{\mathbf{f}} = \underset{\mathbf{f}}{\operatorname{argmin}} L(\mathbf{f}) \tag{20}$$

donde los "parámetros" $f \in R^N$ son los valores de la función de aproximación $f(x_i)$ en cada uno de los N puntos de datos x_i:

$$\mathbf{f} = \{f(x_1), .., f(x_N)\}$$

Los procedimientos de optimización numérica resuelven (20) como una suma de vectores componente

$$\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m,$$

con $h_m \in R^N$

donde $\mathbf{f}_0 = \mathbf{h}_0$ es una conjetura inicial, y cada $\mathbf{f}_m$ sucesivo se induce basándose en el vector de parámetros actual $\mathbf{f}_{m-1}$, que es la suma de las actualizaciones inducidas anteriormente. Los métodos de optimización numérica difieren en sus prescripciones para calcular cada vector de incremento $\mathbf{h}_m$ ("step").

La estrategia de Steepest descent es elegir $\mathbf{h}_m$ de tal manera que $\mathbf{h}_m = -\rho_m g_m$ donde ρ_m es un escalar $g_m \in R^N$ es el gradiente de $L(\mathbf{f})$ evaluado en $\mathbf{f} = \mathbf{f}_{m-1}$. los componentes de el gradiente g_m son

$$g_{im} = [\frac{\partial L(y_i, f(x_i)}{\partial f(x_i)}]_{f(x_i)=f_{m-1}(x_i)} \tag{21}$$

la longitud del paso ρ_m es la solución a

$$\rho_m = \underset{\rho}{\operatorname{argmin}} L(\mathbf{f}_{m-1} - \rho g_m) \tag{22}$$

La solución actual es luego actualizada

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m g_m$$

y el proceso se repite en la siguiente iteración. El Steepest Descent puede verse como una estrategia muy exigente, ya que $-g_m$ es la dirección local en R_N para la cual $L(\mathbf{f})$ es más rápidamente decreciente en $\mathbf{f} = \mathbf{f}_{m-1}$.

2.4.1. Algoritmo para Gradient Tree Boost

Ahora, se realizará una analogia con el problema de los arboles descritos en la sección de boosting. El algoritmo 2.3.3 es una estrategia también muy exigente al igual que Steepest Descent para resolver (17). En cada paso el árbol solución es el que reduce al máximo (17), dado el modelo actual f_{m-1} y sus ajustes $f_{m-1}(x_i)$ Así, las predicciones del árbol $T(x_i;\Theta_m)$ son análogas a las componentes del gradiente negativo (21). La principal diferencia entre $T(x_i;\Theta_m)$ y g_{im} es que los componentes del árbol $t_m = (T(x_1;\Theta_m),...,T(x_1;\Theta_m))$ no son independientes. Están restringidos a ser las predicciones de un árbol de decisión de nodos J_m-terminales, mientras que el gradiente negativo es la dirección de descenso máxima no restringida.

La solución (18) usando la aproximaxión "Forward stagewise" es analoga a el enfoque de (22) en steepest descent. La diferencia es que (18) realiza su enfoque separada de aquellas componentes de $\mathbf{t}_m$ que corresponden a cada región terminal separada $\{T(x_i;\Theta_m)\}_{x_i\}enR_{jm}}$.

Si el único objetivo fuera minimizar la pérdida en los datos de entrenamiento (19), steepest descent sería la estrategia preferida. El gradiente (21) se puede calcular para cualquier función de pérdida diferenciable $L(y;f(x))$, mientras que resolver (17) es difícil para los criterios mas robustos. Desafortunadamente, el gradiente (21) se define sólo en los puntos de datos de entrenamiento x_i , mientras que el objetivo final es generalizar el modelo de árbol potenciado $f_M x)$ a nuevos datos no representados en el conjunto de entrenamiento.

Una posible resolución a este dilema es inducir un árbol $T(x;\Theta_m)$ en la m-ésima iteración cuyas predicciones $\mathbf{t}_m$ estén lo más cerca posible del gradiente negativo. Utilizando el error al cuadrado para medir la cercanía, esto nos lleva a

$$\tilde{\Theta}_m = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{N} (-g_{im} - T(x_i;\Theta))^2 \tag{23}$$

Es decir, se ajusta el árbol T a los valores de gradiente negativo (21) por mínimos cuadrados. Aunque las regiones solución $\tilde{R}_{jm}$ a (23) no serán idénticas a las regiones R_{jm} que resuelven (17), en general es lo suficientemente similar como para servir al mismo propósito. En cualquier caso, el procedimiento "forward stagewise boosting", y la inducción de árbol de decisión, son en sí mismos procedimientos de aproximación. Después de construir el árbol (23), las constantes correspondientes en cada región vienen dadas por (18).

Para una clasificación de K clases donde la respuesta Y toma valores en el conjunto no ordenado $G = \{G_1, ..., G_k\}$ y las probabilidades condicionales de clase está dada por $p_k(x) = Pr(Y = G_k|x), k = 1, 2, ..., K$, la función de pérdida el gradiente negativo es simplemente la desviación multinomial

$$L(y, p(x)) = -\sum_{k=1}^{K} I(y = G_k) log p_k(x) = -\sum_{k=1}^{K} I(y = G_k) f_k(x) + log(\sum_{l=1}^{K} e^{f_l(x)}) \tag{24}$$

y se construyen K árboles de mínimos cuadrados en cada iteración. Cada árbol T_{km} se ajusta a su respectivo vector de gradiente negativo g_{km},

$$-g_{ikm} = \frac{dL(y_i, f_{1m}(x_i), ..., f_{1m}(x_i))}{df_{km}(x_i)} = I(y_i = G_k) - p_k(x_i) \tag{25}$$

con $p_k(x)$ dado por

$$\frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}} \tag{26}$$

Aunque se construyen K árboles distintos en cada iteración, están relacionados a través de (26)

Ahora se describirá de manera mas explicita el algoritmo Gradient tree Boosting

- Incializa $f_0(x) = \underset{\gamma}{\text{argmin}} \sum_{i=1}^{N} L(y_i, \gamma)$
- Para $m = 1$ a M:
  - a Para $i = 1, 2, .., N$ Calcula
  $$r_{im} = \frac{dL(y_i, f_{1m}(x_i), ..., f_{1m}(x_i))}{df_{km}(x_i)}$$
  - b Ajusta un arbol de clasificación para los objetivos r_{im} obteniendo regiones terminales R_{jm}, $j = 1, 2, ..., J_m$
  - c For $j = 1, 2..., J_m$ calcula
  $$\gamma_{jm} = \underset{\gamma_{jm}}{\text{argmin}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$
  - d Actualiza $f_m = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
- Salida $\hat{f}(x) = f_M(x)$

La primera línea del algoritmo se inicializa en el modelo constante óptimo, que no es más que un árbol de un solo nodo terminal. Los componentes del gradiente negativo calculado en la línea 2(a) se denominan residuos generalizados o pseudo residuos, r. Para la clasificación se toma como gradiente la ecuación (25). Luego, las líneas $2(a)-(d)$ se repiten K veces en cada iteración m, una vez para cada clase. El resultado en la línea 3 son K expansiones de árbol diferentes (acopladas) $f_{kM}(x)$; $k = 1,2,..,K$ Estos expansiones de arboles hacen la clasificación con el siguiente clasificador bayesiano

$$G(x) = G_k \tag{27}$$

Donde

$$k = \underset{l}{\operatorname{argmax}} p_l(x)$$

y

$$p_k(x) = P(Y = G_k|x)$$

para $k = 1,2,..,K$ Dos parámetros básicos de ajuste son el número de iteraciones M y los tamaños de cada uno de los árboles constituyentes $J_m, m = 1,2,...,M$.

2.5. Importancia de las variables predictoras

Para un único árbol de decisión T, Breiman et al. (1984) propusieron

$$\Upsilon_l^2(T) = \sum_{t=1}^{J-1} i_t^2 I(v(t) = l)$$

como medida de relevancia para cada variable predictora X_l. La suma es sobre los $J-1$ nodos internos del árbol. En cada nodo t, se utiliza una de las variables de entrada $X_{v(t)}$ para dividir la región asociada a ese nodo en dos subregiones; dentro de cada una, se aplica una constante distinta a los valores de respuesta. La variable particular elegida es la que da la máxima mejora estimada i_t^2 en el riesgo de error al cuadrado sobre la de un ajuste constante en toda la región. La importancia relativa al cuadrado de la variable X_l es la suma de dichas mejoras al cuadrado en todos los nodos internos para los que se eligió como variable de división.

Esta medida de importancia se generaliza fácilmente a las expansiones aditivas de árboles (16); simplemente se promedia sobre los árboles

$$\Upsilon_l^2 = \frac{1}{M} \sum_{m=1}^{M} \Upsilon_l^2(T_m) \tag{28}$$

Para clasificación de K-Clases, K modelos separados $f_k(x); k = 1,2,...,K$ son inducidos, cada uno formado por una suma de árboles

$$f_k(x) = \sum_{m=1}^{M} T_{km}(x)$$

En este caso (14.43) generalizes to

$$\Upsilon_{lk}^2 = \frac{1}{M} \sum_{m=1}^{M} \Upsilon_l^2(T_{km})$$

Aquí Υ_{lk} es la relevancia de X_l en la separación de las observaciones de la clase k de de las demás clases. La relevancia global de X_l se obtiene promediando todas las clases

$$\Upsilon_l^2 = \frac{1}{K}\sum_{m=1}^{K}\Upsilon_{lK}^2$$

Ademas de esta medida, existen otras maneras de medir la importancia de las variables las cuales son:

2.6. Metodo de la transformada inversa

Supongamos que se desea generar el valor de una variable aleatorioa discreta X, con función de masa de probabilidad

$$P\{X = x_j\} = p_j, j = 0,1,...,\sum_j p_j = 1$$

Como $P\{a \leq U < b\} = b - a$ para $0 < a < b < 1$, tenemos que

$$P\{X = x_j\} = P\{\sum_j^{j-1} p_j \leq U < \sum_j^{j} p_j\} = p_j$$

De manera sintetizada, este método consiste en gener un numero aleatorio U distribuido uniforme $(0,1)$ para así determinar el valor de nuestra variable aleatoria deseada en el intervalor $F(x_{j-1}), F(x_j)$ en el que está U o en poca palabras, hallar la inversa de $F(U)$

2.6.1. Algoritmo método transformada inversa

- Generar un número aleatorio U
- Si $U < p_0$, hacer $X = x_0$ y terminar
- Si $U < p_0 + p_1$, hacer $X = x_1$ y terminar
- Si $U < p_0 + p_1 + p_2$, hacer $X = x_2$ y terminar . . .

Si los x_i con $i >= 0$ están ordenados de tal manera que $x_0 < x_1 < x_2 <$ ..... y si F denota la función de distribución de X, entonces $F(X) = \sum_{i=0}^{k} p_i$ y entonces X será igual a x_j si $F(x_{j-1}), F(x_j)$

2.7. Importancia por permutación

Podemos determinar la influencia de cada predictor en una métrica de evaluación específica del modelo (como el error out-of-bag o la validación cruzada) de la siguiente manera:

- Crear el conjunto de árboles que forman el modelo.
- Calcular una determinada métrica de error (mse, classification error, ...). Este es el valor de referencia ($error_0$).

- Para cada predictor j
    - Permutar en todos los árboles del modelo los valores del predictor j manteniendo el resto constante.
    - Recalcular la métrica tras la permutación, llámese error_j.
    - Calcular el incremento en la métrica debido a la permutación del predictor j.

$$\text{incremento}_j = \frac{\text{error}_j - \text{error}_0}{\text{error}_0} * 100$$

Si un predictor se encuentra contribuyendo al modelo, es de esperar que al permutarlo se produzca un aumento en el error del modelo. Esto se debe a que se pierde la información que esa variable proporcionaba anteriormente. El grado en que el error aumenta debido a la permutación del predictor j puede interpretarse como la influencia que dicho predictor tiene sobre el modelo.

Sin embargo, es importante tener en cuenta que este incremento en el error puede resultar negativo, lo cual puede generar confusión. Si la variable no estaba contribuyendo significativamente al modelo, es posible que al reorganizarla aleatoriamente se obtenga una mejora leve en el modelo por pura casualidad. Por lo tanto, la diferencia $(\text{error}_j - \text{error}_0)$ podría ser negativa.

En términos generales, cuando este incremento en el error es negativo, se puede considerar que dichas variables tienen una importancia cercana a cero, ya que su permutación aleatoria no afecta significativamente al modelo.

2.7.1. Incremento de la pureza de nodos

Cuantifica el incremento total en la pureza de los nodos debido a divisiones en las que participa el predictor (promedio de todos los árboles). La forma de calcularlo es la siguiente: en cada división de los árboles, se registra el descenso conseguido en la medida empleada como criterio de división (índice Gini, mse entropía, ...). Para cada uno de los predictores, se calcula el descenso medio conseguido en el conjunto de árboles que forman el ensemble. Cuanto mayor sea este valor medio, mayor la contribución del predictor en el modelo.

2.8. Grid Search

Los modelos de machine learning están compuestos por parámetros e hiperparamtros. Los parámetros son los coeficientes internos que los modelos obtienen al momento de ser entrenados con nuestros datos, por lo que la existencia de estos parametrós está dada por los datos, en cambio los hiperparametros son valores que el modelo no obtiene automaticamente, sino que el usuario es el encargado de introducirlos manualmente, por lo que los valores que estos hiperparametros dependen del problema y la experticia de las personas involucradas en el desarrollo del modelo. Normalmente, un hiperparámetro tiene un efecto conocido sobre un modelo en sentido general, pero no está claro cuál es la mejor manera de establecer un hiperparámetro para un conjunto de datos determinado. Además, muchos modelos de aprendizaje automático tienen una serie de hiperparámetros que pueden interactuar de forma no lineal.

2.9. Metricas

Las metricas que se usan para medir la calidad de un modelo dependen de la naturaleza del modelo. Como este problema es puramente de clasificación, es necesario conocer que tan preciso es un modelo al

momento de clasificar y además de eso que tanto aprendió de los datos, asi que se usaron las siguientes métricas

- **Precisión** la precisión de una clase es el número de verdaderos positivos (es decir, el número de elementos etiquetados correctamente como pertenecientes a la clase positiva o **tp**) dividido por el número total de elementos etiquetados como pertenecientes a la clase positiva (es decir, la suma de los verdaderos positivos y los falsos positivos o **fp**, que son los elementos etiquetados incorrectamente como pertenecientes a la clase).

$$P = \frac{t_p}{t_p + f_p} \tag{29}$$

- **Recall** El recall se define como el número de verdaderos positivos dividido por el número total de elementos que pertenecen realmente a la clase positiva (es decir, la suma de los verdaderos positivos y los falsos negativos o **fn**, que son elementos que no se etiquetaron como pertenecientes a la clase positiva pero deberían haberlo sido).

$$R = \frac{t_p}{t_p + f_n} \tag{30}$$

- **F1 Score** F1 score commbina las medidas de precisión y exhaustividad para devolver una medida de calidad más general del modelo

  The F score is the harmonic mean of the precision and recall. It thus symmetrically represents both precision and recall in one metric.

$$F = \frac{2(R \times P)}{P + R} \tag{31}$$

- **Matriz de confusión**

  Una matriz de confusión, también conocida como matriz de error, es una tabla específica que permite visualizar el rendimiento de un algoritmo, normalmente de aprendizaje supervisado. Cada fila de la matriz representa las instancias de una clase real, mientras que cada columna representa las instancias de una clase predicha, o viceversa (ambas variantes se encuentran en la bibliografía). El nombre se debe a que permite ver fácilmente si el sistema confunde dos clases (es decir, si suele etiquetar erróneamente una como otra).

  En la figura 5 se puede observar un ejemplo de una matriz de confusión de un modelo que trabaja con 4 clases las cuales son 1, 2,3 y 5 y en donde el eje x indica las clases que predijo el modelo y el eje y la verdaderas clases. Para leer adecuadamente la matriz de confusión podemos considerar la diagonal como los aciertos del modelo y como se puede observar, el modelo en la clase 2 no acertó en ninguna predicción prediciendo clase 5 en vez de clase 2 en 4 observaciones de la clase 2, siendo esto una çonfusión". Similarmente 6 observacoi

2.10. Curvas ROC

Una curva ROC (Receiver Operating Characteristic) es una herramienta de evaluación utilizada en el análisis de clasificación y en la evaluación de modelos predictivos. Proporciona una representación visual de la capacidad de un modelo para distinguir entre clases o categorías.

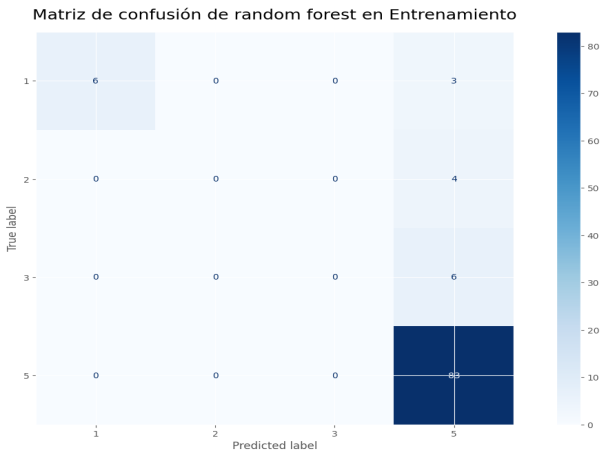


Figura 5: Matriz de confusión

La curva ROC se crea trazando la tasa de verdaderos positivos (Sensibilidad) en el eje Y y la tasa de falsos positivos (1 - Especificidad) en el eje X. Para entender cómo se interpreta, primero necesitamos comprender algunos conceptos clave:

- Tasa de verdaderos positivos (Sensibilidad): También conocida como tasa de detección o recall, representa la proporción de casos positivos que se clasifican correctamente como positivos en relación con el total de casos positivos reales.
- Tasa de falsos positivos (1 - Especificidad): Es la proporción de casos negativos que se clasifican incorrectamente como positivos en relación con el total de casos negativos reales. Se representa como 1 - Especificidad porque la especificidad es la proporción de casos negativos correctamente clasificados.

Cuando se representa una curva ROC, cada punto en la curva corresponde a un umbral de clasificación diferente. El umbral determina el punto de corte para decidir si una instancia se clasifica como positiva o negativa. Al variar el umbral, se obtienen diferentes pares de sensibilidad y especificidad, lo que da como resultado diferentes puntos en la curva.

Idealmente, deseamos que la curva ROC se acerque lo más posible al rincón superior izquierdo del gráfico como se observa en la figura 6, lo que indicaría una alta sensibilidad (todos los casos positivos se clasifican correctamente) y una baja tasa de falsos positivos (se clasifican pocos casos negativos como positivos). En este caso, el área bajo la curva ROC (AUC-ROC) será cercana a 1, lo que indica un modelo con un buen rendimiento.

Una curva ROC también puede proporcionar información sobre el rendimiento relativo de diferentes modelos. Si se superponen varias curvas ROC en el mismo gráfico, se puede comparar visualmente la capacidad de discriminación de cada modelo. El modelo que tenga una curva ROC más cercana al rincón superior izquierdo generalmente se considera el mejor en términos de rendimiento.

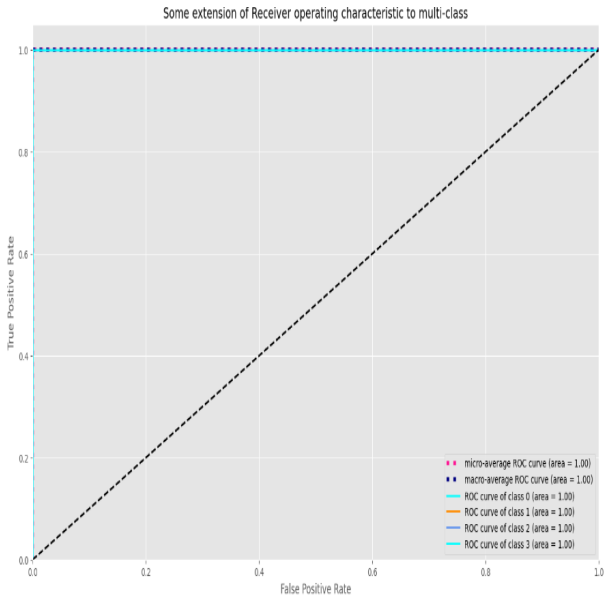


Figura 6: Ejemplo curva ROC

3. Metodología

3.1. Identificación del problema

Incialmente se plantea un problema de calificación para el estado de los transformadores de corriente sin TAP antiguos a partir de un conjunto de reglas pre-establecidas por los expertos y provedores de equipos. Un transformador de corriente funciona para aumentar o disminuir una corriente alterna y es importante para aíslar y proteger los equipos de medida y protección de los altos niveles de tensión del sistema primario. Los equipos a lo largo del tiempo sufren degradación debido a condiciones externas como el clima, ingreso de animales, o el tiempo transcurrido de funcionamiento, asi que no solamente se tiene en cuenta los valores medidos en el momento sino además su comportamiento historico. Ahora, Las caracterisiticas que fueron previamente decididas para determinar el estado de salud de un transformador de corriente se presentan en la siguiente tabla:

Variable	Tipo de vairable	Tipo de valor
Capacitancia	Cuantitativa	Numeros Reales
Factor de potencia 10kv	Cuantitativa	Numeros Reales
Factor de potencia 10kv	Cuantitativa	Numeros Reales
Medida de resitencia Núcleo	Cuantitativa	Numeros Reales
Severidad Por termografía	Cualitativa	Categórica ordinal
Estado cajetin	Cualitativa	Categórica ordinal
Estado orcelana	Cualitativa	Categórica ordinal
Inspección diafragma	Cualitativa	Categórica ordinal
Nivel de aceite	Cualitativa	Categórica ordinal
Fuga de aceite	Cualitativa	Categórica ordinal
Inspección visual general	Cualitativa	Categórica ordinal

No todas las variables anteriormente enlistadas se les estudia su comportamiento historico. Unicamente se analiza este aspecto en las variables Capacitancia, Factor de potencia 10kv, Factor de potencia 2.5kv. La manera elegida para medir la degradación es con la siguiente formula:

$$\frac{\text{Valor final} - \text{Valor anterior}}{\text{Valor anterior}} \tag{32}$$

Esta formula es alimentada por los dos ultimos registros de los equipos que llevan varias revisiones. Ademas, se tiene en cuenta a que grupo pertenece el equipo, puesto que a pesar de que todos los equipos se califican con el mismo criterio, los valores de decisión pueden variar segun la empresa que fabricó el equipo. Lo anterior no está en el actual algoritmo de calificación y el uso de aprendizaje automatico facilita la implementación de esta regla importante a la hora de un diagnostico adecuado.

ISA intercolombia actualmente califica los transformadores de corriente sin TAP con los valores 1,2,3 y 5. Estos valores poseen las siguientes interpretaciones:

- **Calificación 1:** Este valor indica que el equipo presenta una falla funcional y debe ser retirado para realizar labores de mantenimiento o ser reemplazado en caso de ser necesario
- **Calificación 2:** Este valor indica que el equipo está cerca de sufrir una falla funcional y se debe realizar un plan de mantenimiento de manera inmediata
- **Calificación 3:** Este valor indica que el equipo presenta fallos no tan graves pero que con el transcurso del tiempo pueden llegar a ser fallas funcionales, por lo tanto, se cuestiona el equipo y se prepara un plan de mantenimiento en un tiempo no mayor a 6 meses
- **Calificación 5:** Este valor indica que el equipo está en buen estado

Las calificaciones de los equipos sin importar al grupo al que pertencen los equipos poseen las siguientes reglas:

Variable	Criterio
Capacitancia	Intervalo de cuestionamiento a<x<b
Factor de potencia 10kv	Intervalo de cuestionamiento a<x<b
Factor de potencia 10kv	Intervalo de cuestionamiento a<x<b
Medida de resitencia Núcleo	Intervalo de cuestionamiento a<x<b
Severidad Por termografía	Cal 1 : 1, Cal 2 : 2,Cal 3 : 3
Estado cajetin	Cal 1: Malo, Cal 2 : Regular
Nivel de aceite	Cal 2: Bajo ,Cal 3: Medio
Fuga de aceite	Cal 1:Goteo continuo , Cal 2: Goteo intermitente, Cal 3 : Mancha
Nivel de aceite + fuga de aceite	Cal 1: Medio + Goteo intermitente o continuo, Cal 3 : Bueno + Mancha Cal 2 : Bueno + Goteo y Medio + Mancha
Estado porcelana	Cal 1: Roto + hay nivel o fuga cuestionado ,Cal 2 : Malo , Cal 3: Despicado
Inspección diafragma	Cal 1: Malo, Cal 2 : Regular
Inspección visual general	Cal 2: Malo ,Cal 3: Regular
Inspección visual + Porcelana	Cal 1 : Mala + Rota ,Cal 2 : Despicada + Regular

Las reglas de diagnostico no exigen siempre que todas las variables o un conjunto de variables sean las que presenten anomalias para calificar. El criterio base para determinar que variable posee mayor impacto es que siempre será mas relevante la variable que peor califica, es decir, si hay variables que

podrían calificar como 3 y 2 pero hay al menos una que califica 1, esta ultima será la que definirá la calificación final. Otro aspecto a tener en cuenta es que si alguna de las variables está muy cerca a valores que determinan calificaciones, se prefiere aproximar y cuestionar el equipo, para así prevenir un posible fallo en el futuro. Por ejemplo, supongamos que para cierto grupo de equipos, la capacitancia califica 3 entre los 0,025 y 0,03 y menor a 0,025 considerando el resto de variables en sus valores normales calificaria 5, luego tenemos que el valor de medida marca 0,02498, en este caso, se prefiere calificar 3, puesto que no cuestionar este equipo sería ignorar el hecho de que con el pasar del tiempo este equipo puede empeorar su estado y generar un reporte erroneo , lo que causaria accidentes o una falla fucional grave en el equipo.

Teniendo en cuenta todas las ilustraciones presentadas a la hora de identificar el problema, se concluyó que es un problema de clasificación en el ambito del aprendizaje automatico, por lo tanto, se reduce la cantidad de modelos que puedan ser utiles para evaluar el estado de los equipos. Con las caracteristicas anteriores se tiene un conjunto de criterios para la calificación de cada uno de estos equipos y esto es un reto para ISA puesto que la mala calificación de un equipo puede desencadenar accidentes y fallas en el flujo electrico del pais, generando perdidas millonarias e incluso humanas dado que los accidentes de estos equipos suelen ser explosiones muy peligrosas.

3.2. Data Wrangling

(a) DeepNote (b) Python

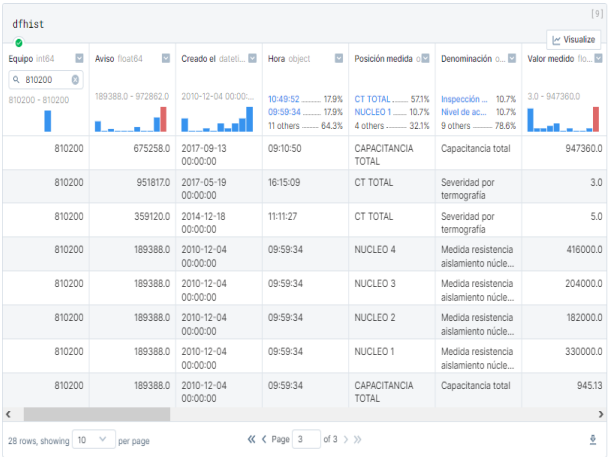
El entorno usado para trabajar fue Deepnote. Es una aplicación web compatible con Jupyter Notebooks que provee de un entorno de trabajo interactivo para ciencia de datos a nivel de proyectos. En él se puede escribir y ejecutar código de una forma sencilla y colaborativa en tiempo real sin ninguna configuración extra. Se puede trabajar con distintas versiones de Python, R, SQL o crear tu propio entorno personalizado, lo que permite tener flexibilidad a la hora de realizar algoritmos para obtener la información que se necesita.

Los datos que son utilizados para las calificaciones de los equipos son extraidos de la plataforma SAP , y son obtenidos de la siguiente forma:

En la tabla (b) de la figura 8, tenemos los datos extraídos de SAP que contienen toda la información que nosotros necesitamos. Las columnas de esta tabla útiles para el problema son:

- Equipo : Esta columna posee el id del equipo en cuestión y nos ayudará a identificar a que grupo de equipos pertencen
- Denominacion : Esta columna posee todos los nombres de las variables que se usaran para calificar
- Valor medido : Esta columna posee los valores de cada variable
- Codif.txt.cód. : Esta columna tiene los perfiles de catalogo de las variables categoricas.
- Creado el: Esta columna nos indica la fecha en la que se realizó la prueba en alguna de las variables.

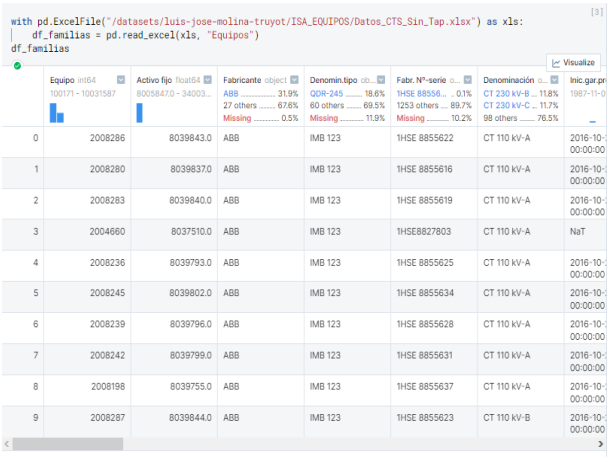
Figura 8: Tablas principales



dfhist

Equipo int64	Aviso float64	Creado el datet...	Hora object	Posición medida o...	Denominación o...	Valor medido flo...
810200	675258.0	2017-09-13 00:00:00	09:10:50	CAPACITANCIA TOTAL	Capacitancia total	947360.0
810200	951817.0	2017-05-19 00:00:00	16:15:09	CT TOTAL	Severidad por termografía	3.0
810200	359120.0	2014-12-18 00:00:00	11:11:27	CT TOTAL	Severidad por termografía	5.0
810200	189388.0	2010-12-04 00:00:00	09:59:34	NUCLEO 4	Medida resistencia aislamiento núcle...	416000.0
810200	189388.0	2010-12-04 00:00:00	09:59:34	NUCLEO 3	Medida resistencia aislamiento núcle...	204000.0
810200	189388.0	2010-12-04 00:00:00	09:59:34	NUCLEO 2	Medida resistencia aislamiento núcle...	182000.0
810200	189388.0	2010-12-04 00:00:00	09:59:34	NUCLEO 1	Medida resistencia aislamiento núcle...	330000.0
810200	189388.0	2010-12-04 00:00:00	09:59:34	CAPACITANCIA TOTAL	Capacitancia total	945.13

(a) Tabla familias



```
with pd.ExcelFile("/datasets/luis-jose-molina-truyot/ISA_EQUIPOS/Datos_CTS_Sin_Tap.xlsx") as xls:
    df_familias = pd.read_excel(xls, "Equipos")
df_familias
```

	Equipo int64	Activo fijo float64	Fabricante object	Denomin.tipo ob...	Fabr. Nº-serie o...	Denominación o...	Inic.gar.pr
0	2008286	8039843.0	ABB	IMB 123	1HSE 8855622	CT 110 kV-A	2016-10-... 00:00:00
1	2008280	8039837.0	ABB	IMB 123	1HSE 8855616	CT 110 kV-A	2016-10-... 00:00:00
2	2008283	8039840.0	ABB	IMB 123	1HSE 8855619	CT 110 kV-A	2016-10-... 00:00:00
3	2004660	8037510.0	ABB	IMB 123	1HSE8827803	CT 110 kV-A	NaT
4	2008236	8039793.0	ABB	IMB 123	1HSE 8855625	CT 110 kV-A	2016-10-... 00:00:00
5	2008245	8039802.0	ABB	IMB 123	1HSE 8855634	CT 110 kV-A	2016-10-... 00:00:00
6	2008239	8039796.0	ABB	IMB 123	1HSE 8855628	CT 110 kV-A	2016-10-... 00:00:00
7	2008242	8039799.0	ABB	IMB 123	1HSE 8855631	CT 110 kV-A	2016-10-... 00:00:00
8	2008198	8039755.0	ABB	IMB 123	1HSE 8855637	CT 110 kV-A	2016-10-... 00:00:00
9	2008287	8039844.0	ABB	IMB 123	1HSE 8855623	CT 110 kV-B	2016-10-... 00:00:00

(b) Tabla Documentos de medida

```
[6]
def clean_col(dataset):
  col_f=list(dataset.columns)
  lista=[]
  for columna in col_f:
    columna=re.sub(r"\s+", "", columna)
    columna= unidecode(columna)

    lista.append(columna)
  diccion=dict(zip(col_f,lista))
  data_nuevo=dataset.rename(diccion, axis=1)
  return data_nuevo
```

```
[7]
def Normalization_tabla(df,columna):
    df[columna] = df[columna].str.strip()
    df[columna] = df[columna].str.lower()
    df[columna] = df[columna].apply(lambda x: x.replace (" ", "_"))
    df[columna] = df[columna].apply(lambda x: unidecode(x))
    return df
```

Figura 9: Normalizacion de nombres de documentos de medida

En la tabla (a) figura 8, podemos ver otra tabla que posee la columna equipo, Familia y subfamilia. Esta tabla permite poder relacionar el id de cada equipo con su familia y subfamilia y así asignarle correctamente a cada transformador el grupo al que pertenece, siendo esto último fundamental a la hora de calificar.

Para la disposición final de los datos en el marco del modelo de clasificación que se usaron, se le realizó el siguiente preprocesamiento de los datos:

- Para la tabla 2, se homologaron los nombres de las familias y subfamilias con el fin de agrupar correctamente los equipos. Para la tabla 1, se normalizaron los nombres de la columna "Denominación"que es la con contiene el nombre de las variables. Esto se hizo con el fin de evitar columnas redundantes o tener que realizar más tareas de procesamiento después. Se pueden observar las funciones creadas para esta labor en la figura 9
- Como el enfoque del problema solo se preocupa por los equipos que poseen más de un registro en sus variables a lo largo del tiempo, se separaron los equipos en nuevos y viejos, los viejos son los equipos que se van a analizar. En la figura 10 (a) se puede observar el ejemplo de un equipo que posee dos valores de Nivel de aceite en varias fechas, lo que indica que este equipo es ”viejo”.

Figura 10

100171	nan	2017-06-28 00:00:00	12:59:24	CT TOTAL	Nivel de aceite	nan	nan
100171	nan	2017-05-24 00:00:00	16:48:05	CT TOTAL	Inspección visual general	nan	nan
100171	nan	2017-05-24 00:00:00	16:48:05	CT TOTAL	Fuga de aceite	nan	nan
100171	nan	2017-05-24 00:00:00	16:48:05	CT TOTAL	Nivel de aceite	nan	nan

(a) Variables con varios registros

```
counts = df_datos_correctos.groupby(["Equipo", "Denominacion"]).size().reset_index(name="num_registros")

# Identificar equipos nuevos y viejos
nuevos = counts.groupby("Equipo")["num_registros"].max() == 1
viejos = counts.groupby("Equipo")["num_registros"].max() > 1

# Crear una nueva columna en el DataFrame original para etiquetar los equipos como nuevos o viejos
df_datos_correctos["estado_equipo"] = "nuevo"
df_datos_correctos.loc[df_datos_correctos["Equipo"].isin(viejos[viejos].index), "estado_equipo"] = "viejo"

# Mostrar la tabla con la nueva columna de estado de equipo
df_datos_correctos
```

	Equipo	Creadoel	Denominacion	Valormedido	estado_equipo
0	100171	2020-11-27 00:00:00	calificacion_del_equipo	3.0	viejo
1	100171	2017-06-28 00:00:00	fuga_de_aceite	No	viejo
2	100171	2017-06-28 00:00:00	inspeccion_visual_general	Bueno	viejo
3	100171	2017-06-28 00:00:00	nivel_de_aceite	Normal	viejo

(b) Clasificación de equipos segun su antiguedad

- En los criterios de calificación se tiene en cuenta el comportamiento histórico de los equipos, por lo tanto, se necesitó separar las variables categóricas de las no-categóricas con el fin de aplicarle la ecuación 32 que nos permita captar la información histórica de las variables capacitancia y los factores de potencia 10kv y 2.5kv. Las medidas de resistencia de aislamiento fueron también separadas puesto que a estas no se le aplicará esta fórmula, se evaluará según los criterios anteriormente definidos.

Figura 11: Tablas principales

Cuantitativos

```
df_viejos_var_cuant = df_historicos[~df_historicos['Denominacion'].isin(documentos_interes)]
df_viejos_var_cuant
```

	Equipo	Creadoel	Denominacion	Valormedido
2	100171	2020-11-24 00:00:00	capacitancia_total	794.65
3	100171	2015-11-04 00:00:00	capacitancia_total	794.98
4	100171	2020-11-24 00:00:00	factor_de_potencia_10kv_ctotal	0.34
5	100171	2015-11-04 00:00:00	factor_de_potencia_10kv_ctotal	0.184
6	100171	2020-11-24 00:00:00	factor_de_potencia_2.5kv_ctotal	0.336

(a) Tabla variables cuantiativas

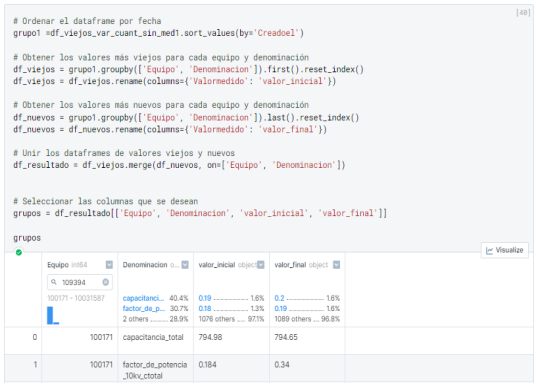
```
medida_resistencia=['medida_resistencia_aislamiento_nucleo_1','medida_resistencia_aislamiento_nucleo_2','medida
df_viejos_var_cuant_medidas1= df_viejos_var_cuant[df_viejos_var_cuant['Denominacion'].isin(medida_resistencia)]
df_viejos_var_cuant_medidas1
```

	Equipo	Creadoel	Denominacion	Valormedido
12	100171	2020-11-27 00:00:00	medida_resistencia_aislamiento_nucl...	50.0
13	100171	2015-10-28 00:00:00	medida_resistencia_aislamiento_nucl...	174.0
14	100171	2020-11-27 00:00:00	medida_resistencia_aislamiento_nucl...	28.0

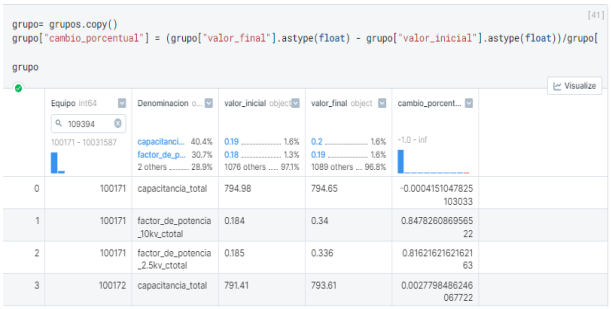
(b) Tabla medidas de resistencia

- Con las tablas debidamente separadas como se puede observar en la **Figura 11**, con ayuda de algoritmos, se obtuvieron los valores medidos finales e iniciales en dos columnas distintas con el fin de realizar los cálculos para obtener el comportamiento historio y obtener el cambio porcentual, el cual nos indica que tanto creció o disminuyó los valores con respecto al tiempo. Para el estudio de estas variables ya no se tiene en cuenta las fechas, por lo tanto, se retiran. Los códigos para realizar esto se pueden observar en la **figura 12**

Figura 12: Valor final y Cambio porcentual



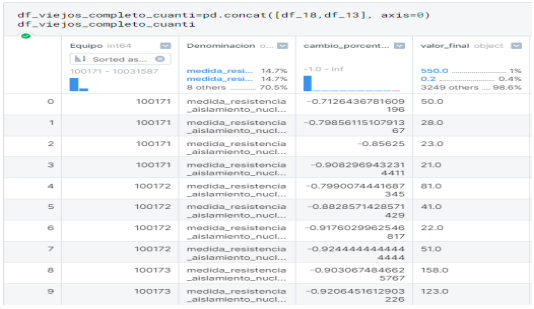
(a) Algoritmo para valor final



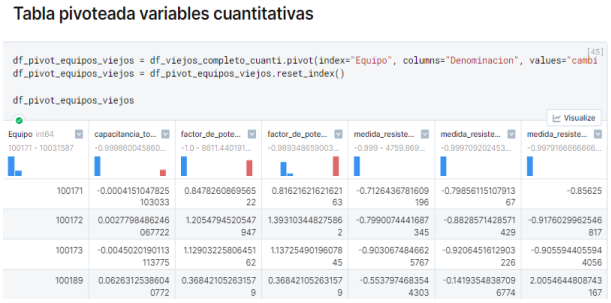
(b) Algoritmo cambio porcentual

- Ya con la tabla con los valores del comportamiento histórico de las variables, se procede a pivotear con las columnas Equipo, denominación y cambio porcentual como en la **figura 13** con el fin de adaptar los datos al formato que necesitan los modelos. La tabla debe tener el id del equipo, las columnas Capacitancia, factor potencia 10,2.5kv con sus valores históricos. El valor final no se usa para pivotear en este caso puesto que genera problemas con la información y filas vacías

Figura 13: Tratamiento variables cuantitativas



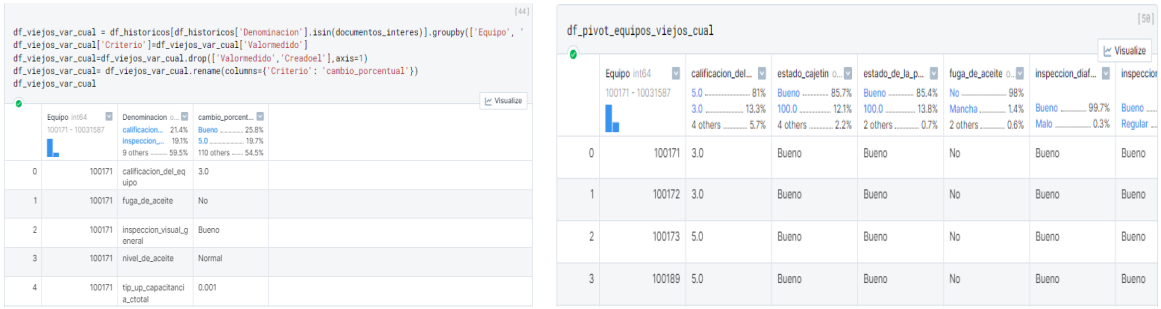
(a) Tabla sin valores



(b) Tabla valores finales

- Para la tabla de variables categóricas se realizó lo mismo que para las otras variables, con la excepción que aquí no se tiene en cuenta el comportamiento histórico de los datos, sino el ultimo valor obtenido en la revisión técnica más reciente. Esto se puede observar en la **figura 14**

Figura 14: Tratamiento variables cualitativas



```
df_viejos_var_cual = df_historicos[df_historicos['Denominacion'].isin(documentos_interes)].groupby(['Equipo', '
df_viejos_var_cual['Criterio']=df_viejos_var_cual['Valormedido']
df_viejos_var_cual=df_viejos_var_cual.drop(['Valormedido','Creadoel'],axis=1)
df_viejos_var_cual= df_viejos_var_cual.rename(columns={'Criterio': 'cambio_porcentual'})
df_viejos_var_cual
```

	Equipo	Denominacion	cambio_porcent...
0	100171	calificacion_del_equipo	3.0
1	100171	fuga_de_aceite	No
2	100171	inspeccion_visual_general	Bueno
3	100171	nivel_de_aceite	Normal
4	100171	tip_up_capacitancia_ctotal	0.001

(a) Valores cualitativos

```
df_pivot_equipos_viejos_cual
```

	Equipo	calificacion_del...	estado_cajetin	estado_de_la_p...	fuga_de_aceite	inspeccion_diaf...	inspeccion
0	100171	3.0	Bueno	Bueno	No	Bueno	Bueno
1	100172	3.0	Bueno	Bueno	No	Bueno	Bueno
2	100173	5.0	Bueno	Bueno	No	Bueno	Bueno
3	100189	5.0	Bueno	Bueno	No	Bueno	Bueno

(b) Tabla pivoteada cualitativas

- Luego de haber obtenido estas tablas, se procede a unir la tablas como en la figura 15 con la función **pd.merge** , que es adecuada para este caso en el que las tablas se relacionan con la columna equipo pero son de diferentes tamaños, evitando así perdida de información o a la creación entradas vacías

Tabla sin valores finales

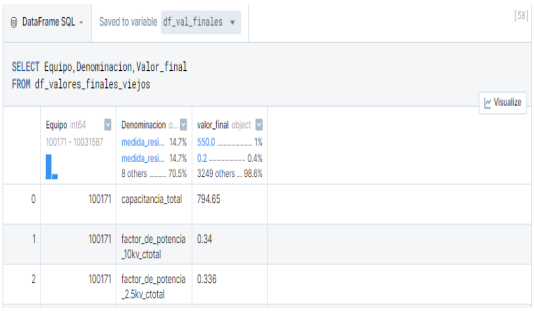
```
tabla_unida = pd.merge(df_pivot_equipos_viejos, df_pivot_equipos_viejos_cual, on='Equipo')
tabla_unida
```

	Equipo	capacitancia_to...	factor_de_pote...	factor_de_pote...	medida_resiste...	medida_resiste...	medida_re
0	100171	-0.000415104782510303	0.847826086956522	0.816216216216263	-0.7126436781609196	-0.7985611510791367	
1	100172	0.0027798486246067722	1.2054794520547947	1.393103448275862	-0.7990074441687345	-0.8828571428571429	-0.9176
2	100173	-0.0045020190113113775	1.1290322580645162	1.137254901960784	-0.9030674846625767	-0.9206451612903226	-0.9055
3	100189	0.06263125386040772	0.3684210526315793	0.3684210526315793	-0.5537974683544303	-0.14193548387096774	2.00546
4	100190	0.06435006435006435	0.7368421052631583	0.7368421052631583	-0.18546365914786966	-0.72109375	-0.6363

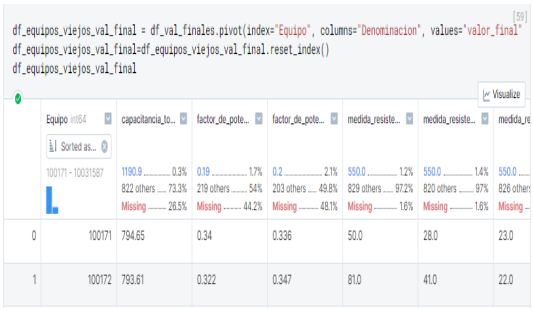
Figura 15: Variables cualitativas unido con la variables cuantitativas

- Para adjuntar los valores finales y los valores de las medidas de resistencia aislamiento a la tabla anterior, primero se concatenan estas dos tablas y luego se aplica la función pivot de Python fijando la columna equipos, como columnas se consideran los nombres de las variables de la columna denominación y los valores medidos pasan a ser los valores de cada columna con respecto a sus equipos. El resultado de esta acción se puede observar en la **figura 16 (b)**

Figura 16: Valores finales sin pivotear



(a) Valores cualitativos



(b) Tabla pivoteada valores finales

- Para diferenciar las columnas que poseen los valores que muestran el comportamiento histórico de las columnas que poseen los valores correspondientes a los últimas pruebas de técnicas realizadas en campo, se les agrega ”_ultimo” al final de los nombres de las columnas que correspondan a estos últimos. Este proceso se puede observar en la **figura 17**

Cambiar nombre de la columna final

```
# Creamos un diccionario con los nuevos nombres de las columnas
new_names = {'capacitancia_total': 'capacitancia_total_ultimo', 'factor_de_potencia_2.5kv_ctotal': 'factor_de_p
 'medida_resistencia_aislamiento_nucleo_2': 'medida_resistencia_aislamiento_nucleo_2_ultimo',
  'medida_resistencia_aislamiento_nucleo_3': 'medida_resistencia_aislamiento_nucleo_3_ultimo',
 'medida_resistencia_aislamiento_nucleo_4': 'medida_resistencia_aislamiento_nucleo_4_ultimo',
 'medida_resistencia_aislamiento_nucleo_5': 'medida_resistencia_aislamiento_nucleo_5_ultimo',
 'medida_resistencia_aislamiento_nucleo_6': 'medida_resistencia_aislamiento_nucleo_6_ultimo',
 'tip_up_capacitancia_total': 'tip_up_capacitancia_total_ultimo'}

# Renombramos las columnas utilizando el método rename() y el diccionario
df_equipos_viejos_val_final = df_equipos_viejos_val_final.rename(columns=new_names)
df_equipos_viejos_val_final
```

Visualize

	Equipo int64 100171 - 10031587	capacitancia_to... 1190.9 0.3% 822 others 73.3% Missing 26.5%	factor_de_pote... 0.19 1.7% 219 others 54% Missing 44.2%	factor_de_pote... 0.2 2.1% 203 others 49.8% Missing 48.1%	medida_resiste... 550.0 1.2% 829 others 97.2% Missing 1.6%	medida_resiste... 550.0 1.4% 820 others 97% Missing 1.6%	medida_re... 550.0 826 others Missing
0	100171	794.65	0.34	0.336	50.0	28.0	23.0
1	100172	793.61	0.322	0.347	81.0	41.0	22.0
2	100173	793.83	0.33	0.327	158.0	123.0	135.0

Figura 17: Cambio nombre de columnas

- Como podemos observar en la **figura 18**, usando SQL uniremos estos datos con la tabla creada con los valores cualitativos y registros históricos

DataFrame SQL · Saved to variable df_11 [62]

```
SELECT e.*, f.*
FROM df_equipos_viejos_val_final e
LEFT JOIN  tabla_unida f ON e.Equipo = f.Equipo
```

Visualize

	Equipo int64 Sorted as... 100171 - 10031587	capacitancia_to... 1190.9 0.3% 822 others 73.3% Missing 26.5%	factor_de_pote... 0.19 1.7% 219 others 54% Missing 44.2%	factor_de_pote... 0.2 2.1% 203 others 49.8% Missing 48.1%	medida_resiste... 550.0 1.2% 829 others 97.2% Missing 1.6%	medida_resiste... 550.0 1.4% 820 others 97% Missing 1.6%	medida_re 550.0 826 others Missing
0	100172	793.61	0.322	0.347	81.0	41.0	22.0
1	100173	793.83	0.33	0.327	158.0	123.0	135.0
2	100189	860.2	0.26	0.26	705.0	665.0	550.0
3	100190	827.0	0.33	0.33	650.0	357.0	480.0

Figura 18: Tabla con valores finales y valores historicos

- Ahora, con la tabla casi consolidada le agregaremos las familias y subfamilias con ayuda de la tabla de la **figura 8 (b)** y la siguiente query de SQL

DataFrame SQL · Saved to variable df_equipos_hist_final [63]

```
SELECT e.*, f.Fabricante, f.Denomintipo
FROM df_11 e
LEFT JOIN df_fam f ON e.Equipo = f.Equipo
```

Visualize

ja_de_aceite o... 98.1% ncha 1.2% thers 0.7%	inspeccion_diaf... Bueno 99.7% Malo 0.3%	inspeccion_visu... Bueno 99.2% Regular 0.8%	nivel_de_aceite o... Normal 99.7% No aplica 0.2% 2 others 0.2%	severidad_por_t... 5 86.9% 5.0 11.9% 2 others 1.2%	Fabricante object ABB 34.6% GEC ALST... 23.9% 15 others 41.5%	Denomintipo obj... IMB245 24% 46 others 74.8% Missing 1.2%
	Bueno	Bueno	Normal	5	RITZ	OSKF 245.32
	Bueno	Bueno	Normal	5	RITZ	OSKF 245.32
	Bueno	Bueno	Normal	5	RITZ	OSKF 245.32
	Bueno	Bueno	Normal	5	RITZ	OSKF 245.32

Figura 19: Tabla con familias y subfamilias

- La extracción de datos en SAP no es muy eficiente, por lo tanto, aparecen equipos que no pertenecen a los transformadores de corriente sin TAP, así que toca eliminarlos como en la **figura 20**.

```
[65]
df_equipos_hist_final=df_equipos_hist_final[~df_equipos_hist_final['Equipo'].isin([132382,
132383,
132384,
132385,
132386,
132387,
132388,
132389,
132390,
132391,
132392,
132393,
132394,
```

Figura 20: Eliminar equipos

3.2.1. Manejo de datos faltantes

Los datos faltantes en este problema poseen un contexto que depende de las variables que se estén analizando, Por lo que las medidas remediales para datos faltantes pueden variar mucho dependiendo de la variable. Se describirán las situaciones que se encontraron con sus respectivas soluciones

- ? La ausencia de datos en las variables categóricas significa que los valores medidos se asumen como correctos, por ejemplo, un valor faltante en el estado de cajetín permite asumir que su estado es ”Bueno” así que se reemplazaron los valores faltantes de esta tabla con los valores normales medidos que le corresponden a cada variable.

Figura 21: Valores finales sin pivotear

inspeccion_diaf...	inspeccion_visu...	nivel_de_aceite ...	nivel_de_aceite...	nivel_de_aceite...	severidad_por_t...	tip_up_capacita...
Bueno 47.5% Malo 1.7% Missing 50.8%	Bueno 100%	Normal 99.4% Missing 0.6%	Missing 100%	Missing 100%	5.0 11.6% 3.0 0.6% Missing 87.8%	0.009 5% 33 others 33.1% Missing 61.9%
Bueno	Bueno	Normal	nan	nan	nan	nan
Bueno	Bueno	Normal	nan	nan	nan	0.012
nan	Bueno	Normal	nan	nan	nan	nan
nan	Bueno	Normal	nan	nan	nan	nan
nan	Bueno	Normal	nan	nan	nan	nan

(a) Datos faltantes variables categóricas

```
[47]
df_pivot_equipos_viejos_cual['estado_cajetin'].replace(np.nan, 'Bueno', inplace=True)
df_pivot_equipos_viejos_cual['estado_cajetin'].replace(100, 'Bueno', inplace=True)
df_pivot_equipos_viejos_cual['estado_cajetin'].replace(99, 'Bueno', inplace=True)
df_pivot_equipos_viejos_cual['estado_cajetin'].replace(95, 'Bueno', inplace=True)
df_pivot_equipos_viejos_cual['estado_cajetin'].replace(90, 'Bueno', inplace=True)
df_pivot_equipos_viejos_cual['estado_cajetin'].replace(0, 'Malo', inplace=True)
df_pivot_equipos_viejos_cual['estado_de_la_porcelana'].replace(100, 'Bueno', inplace=True)
df_pivot_equipos_viejos_cual['estado_de_la_porcelana'].replace(99, 'Bueno', inplace=True)
df_pivot_equipos_viejos_cual['estado_de_la_porcelana'].replace(0, 'Roto', inplace=True)
df_pivot_equipos_viejos_cual['estado_de_la_porcelana'].replace(np.nan, 'Bueno', inplace=True)
df_pivot_equipos_viejos_cual['fuga_de_aceite'].replace(np.nan, "No", inplace=True)
df_pivot_equipos_viejos_cual['inspeccion_diafragma'].replace(np.nan, "Bueno", inplace=True)
df_pivot_equipos_viejos_cual['inspeccion_visual_general'].replace(np.nan, "Bueno", inplace=True)
df_pivot_equipos_viejos_cual['nivel_de_aceite'].replace(np.nan, "Normal", inplace=True)
df_pivot_equipos_viejos_cual['severidad_por_termografia'].replace(np.nan, 5, inplace=True)
df_pivot_equipos_viejos_cual=df_pivot_equipos_viejos_cual.drop(['nivel_de_aceite_-_no_usar', 'nivel_de_aceite_n
```

(b) Algoritmo para quitar espacios vacios

- En el caso de las medidas de resistencia aislamiento, los datos faltantes representan las siguientes situaciones:

  a No todos los equipos tienen 6 nucleo, habrá equipos con 5 4 o 3 o incluso dos unicos nucleos.

  b Si un equipo tiene n nucleos, el equipo debe tener los nucleos anteriores obligatoriamente, por ejemplo, si un equipo tiene 3 nucleos, debe tener los nucleos 1 2 y 3, pero no puede pasar que tenga el nucleo 1 3 y 5, por lo tanto si se presentan datos faltantes en los nucleos de tal manera que muestran este tipo de casos, se procede a eliminar el equipo.

  c Para el caso en el que los datos faltantes correspondan unicamente a la ausencia de nucleos pero se preseverla la secuencia , se le agrega ”No_tiene” , que indica que no posee el nucleo pero esto solo se hará para los núcleos 3,4,5 y 6 puesto que los transformadores tienen

minimo dos nucleos asi que la ausencia de núcleo 1 o 2 se solucionará eliminando el equipo.Luego para el modelo se codificará con algún que indique que la ausencia del núcleo como en la **Figura 22**

```
[71]
tabla_unida['medida_resistencia_aislamiento_nucleo_5'].replace(np.nan,"No_tiene" ,inplace=True)
tabla_unida['medida_resistencia_aislamiento_nucleo_6'].replace(np.nan,"No_tiene" ,inplace=True)
tabla_unida['medida_resistencia_aislamiento_nucleo_4'].replace(np.nan,"No_tiene",inplace=True)
tabla_unida['medida_resistencia_aislamiento_nucleo_3'].replace(np.nan,"No_tiene" ,inplace=True)

tabla_unida

Visualize
```

Figura 22: Llenar datos faltantes en los núcleos

- ? Los equipos que tengan valores faltantes en las variables Factor de potencia 10kv,2.5kv y capacitancia serán eliminados porque no poseen la suficiente información para ser evaluados. El código usado para esto se puede observar en la **figura 23** (arreglar pie de imagen)

```
[67]
df_equipos_hist_final = df_equipos_hist_final.dropna(subset=['factor_de_potencia_10kv_ctotal',
'factor_de_potencia_2.5kv_ctotal','capacitancia_total'])
```

Figura 23: Llenar datos faltantes en los núcleos

Finalmente, luego de solucionar todos los problemas de datos faltantes, se obtuvo una tabla bastante reducida con respecto a la tabla sin manipular y además con el formato adecuado para los modelos que se van a probar.

calificacio	capacitani	factor_de	factor_de	capacitan	factor_de	factor_de	medida_r	medida_r	medida_r	medida_r	medida_r	medida_r	estado_ca	estado_d	fuga_de_	inspeccio	inspeccio	nivel_de_	severidad	Fabricant
5	0,01700095	1,15041862	0,9291185	1240	0,32088835	0,16985171	6817	5588	8339	9004	934	528	Bueno	Bueno	No	Bueno	Bueno	Normal	5	3
5	0,01115518	0,62929416	1,13735164	816	0,15886155	0,23743234	1645	7462	9593	9645	888	796	Bueno	Bueno	No	Bueno	Bueno	Normal	5	3
5	0,01249259	1,17609818	1,10003156	1340	0,33847683	0,1872204	2849	7704	8730	9855	3105	4485	Bueno	Bueno	No	Bueno	Bueno	Normal	5	3
5	0,01316023	1,01046518	0,55866112	1235	0,1137269	0,07748688	6364	8056	3797	5669	2510	8899	Bueno	Bueno	No	Bueno	Bueno	Normal	5	3

Figura 24: Histograma calificaciones

En la **Figura 24** podemos observar como como debe ser el formato de cualquier dato después del preprocesamiento y antes del ingreso al modelo para la calificación.

3.3. Analisis de datos

3.3.1. Variables Categóricas

Antes de realizar este análisis, se requiere codificar las variables categóricas con el fin de graficar adecuadamente los histogramas de frecuencia que nos ayudaran a obtener conclusiones.

En la **figura 25** podemos observar las variables a las cuales se les aplicará sus respectivas codificaciones y el código usado para esta tarea.

```
[4]
df_sai_copy1["CHANGED_estado_cajetin"] = df_sai_copy1["estado_cajetin"].map({'Malo':1,
                                                                             'Regular':3,
                                                                             'Bueno':5})

df_sai_copy1["CHANGED_estado_porcelana"] = df_sai_copy1["estado_de_la_porcelana"].map({'Roto':1,
                                                                                       'Malo':1,
                                                                                       'Despicado':3,
                                                                                       'Despicada':3,
                                                                                       'Bueno':5})

df_sai_copy1["CHANGED_fuga_de_aceite"] = df_sai_copy1["fuga_de_aceite"].map({'No':5,
                                                                             'Mancha':3,
                                                                             'Goteo intermitente':1,
                                                                             'No hay':5,
                                                                             'goteo intermitente':1,
                                                                             'goteo continuo':0})

df_sai_copy1["CHANGED_inspeccion_diafragma"] = df_sai_copy1["inspeccion_diafragma"].map({'Bueno':5,
                                                                                         'Regular':3,
                                                                                         'Malo':1})

df_sai_copy1["CHANGED_inspeccion_visual_general"] = df_sai_copy1["inspeccion_visual_general"].map({'Bueno':5,
                                                                                                   'Regular':3,
                                                                                                   'Malo':1})

df_sai_copy1["CHANGED_nivel_de_aceite"] = df_sai_copy1["nivel_de_aceite"].map({'Bajo':1,
                                                                               'Normal':5,
                                                                               'Medio':3,
                                                                               'Bueno':5,
                                                                               'medio':3})

df_sai_copy1
```

Figura 25: Decodificación de variables categóricas

El primer aspecto para tratar fue conocer la distribución de frecuencias de la variable objetivo, llamada calificación de los datos ya preprocesados

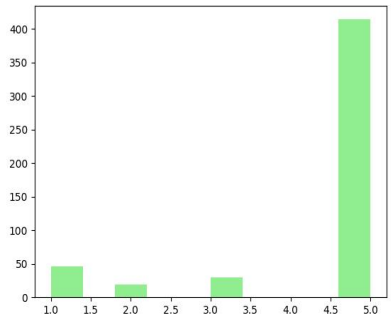


Figura 26: Histograma calificaciones

Se puede observar en la **Figura 26** que la calificación que con más ocurrencias es el 5, por lo tanto se hay una alta probabilidad de que sea necesario crear datos, con el fin de balancear la cantidad de calificaciones y casos sean uniformes.

Ahora, se analizan el resto de las variables respuesta comparándolas con la calificación y estudiar alguna posible relación. Antes de realizar esto, se reasignaron los valores de la siguiente manera.

Estado Cajetin	
Bueno	5
Regular	3
Malo	1

Estado Porcelana	
Bueno	5
Despicado	3
Malo	1

Nivel aceite	
Normal	5
Medio	3
Bajo	1

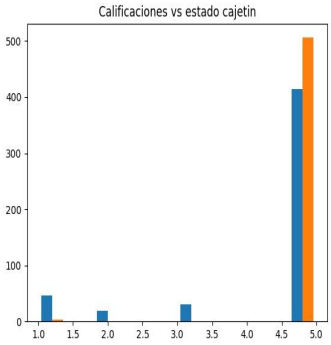
Fuga aceite	
No hay	5
Goteo intermitente	1
Goteo continuo	0

Inspección visual general	
Bueno	5
Regular	3
Malo	1

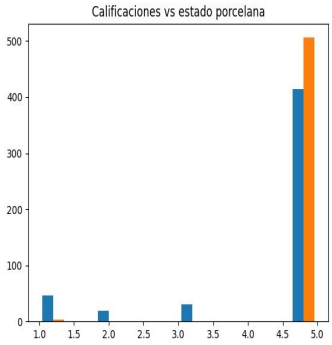
Inspección diafragma	
Bueno	5
Regular	3
Malo	1

Ahora, se enseñarán los diagramas de barras que involucran la variable calificación con el todas las variables categóricas con el fin de ver que tanto se relacionan con respecto a la calificación de los equipos en las **figura 27** y **figura 28**

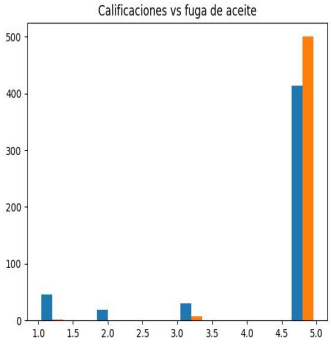
Figura 27: Calificación vs Variables Categóricas



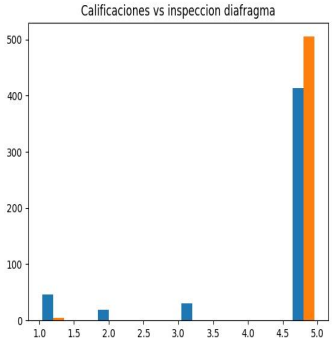
(a) Estado cajetin



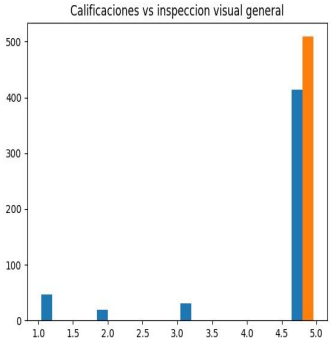
(b) Estado porcelana



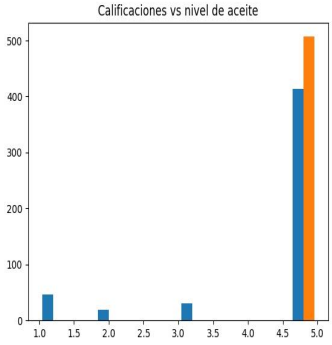
(c) Fuga de aceite



(d) Inspeccion diafragma



(e) Inspeccion visual general



(f) Nivel de aceite

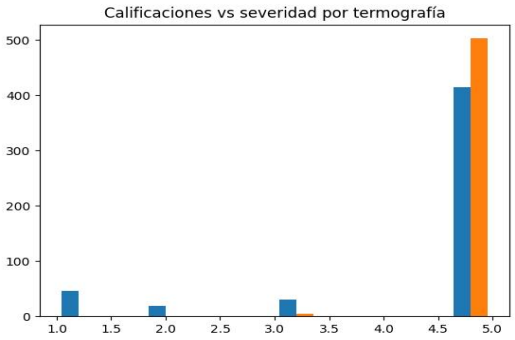


Figura 28: Histograma calificaciones

Se puede observar que, para estos datos las calificaciones se ven poco influidas por estas variables categóricas, por lo tanto, se puede concluir que no hay muchos casos en el que la calificación sea decida por alguna de estas. Este detalle es importante a la hora de realizar un correcto balanceo de datos, puesto que son casos que no se pueden dejar pasar.

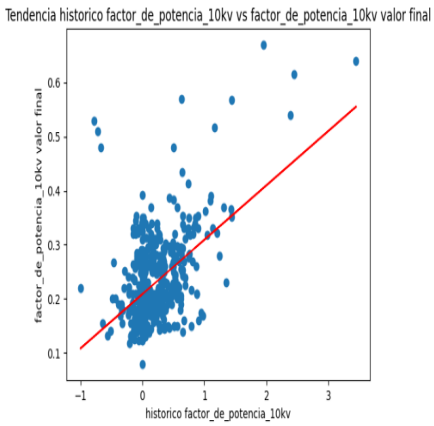
3.3.2. Variables Continuas

Las variables capacitancia, factor de potencia y las medidas de resistencia aislamiento se estudian de manera individual ya que en teoría no poseen ninguna relación a la hora de decidir la calificación, sin embargo, se estudiará las relaciones existentes entre sus valores finales y su comportamiento histórico.

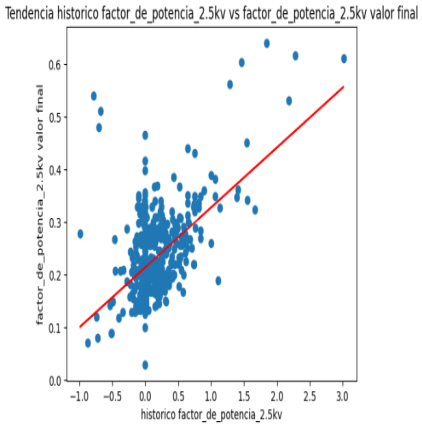
- **Factor de potencia 10kv y 2.5kv**

  Como se puede observar en la figura 29, la relación entre el comportamiento histórico y los últimos valores de medida registrados es claramente creciente. El valor histórico posee valores entre -1 en

Figura 29: Factor de potencia 10kv y 2.5kv vs Comportamiento Historico



(a) Factor de potencia 10kv Ultimo valor vs Comportamiento historico



(b) Factor de potencia 2.5 kv Ultimo valor vs Comportamiento historico

adelante, pero podemos ver acumulación de estos en valores alrededor de 0. Se puede notar que mientras más lejos está de 0, los valores del factor de potencia se sobrepasan el 0,3. Según la información brindada por los expertos en **CTS** los valores normales del factor de potencia tanto para 10kv como 2.5kv están entre 0 y 0,4, y se puede notar que estos umbrales solo son traspasados cuando el comportamiento histórico aumenta y roza valores mayores o iguales a 1, los cuales son valores que ya entran a cuestionar los equipos.

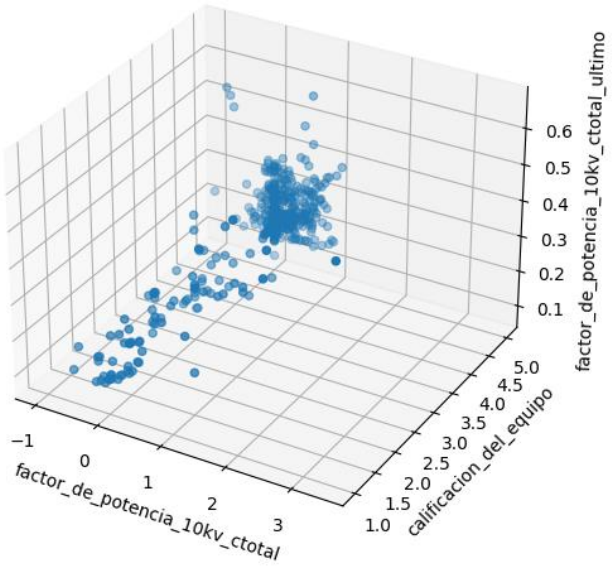
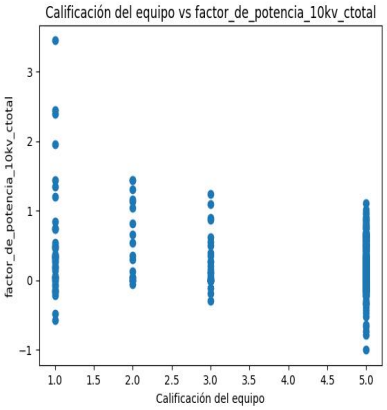


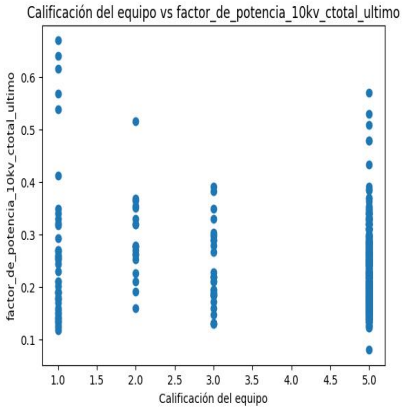
Figura 30: Calificación vs Factor de potencia 10kv vs Comportamiento historico

Se puede observar en la **Figura 30** que los equipos que se califican 5, son por lo general equipos que poseen valores en su comportamiento histórico entre -1 y 1 y en su último valor de medida entre 0 y 0,45, corroborando la información brindada por los expertos en las reuniones. También podemos notar que la mayor concentración de equipos calificados en 5 está entre 0 y 0,45 y para el resto de las calificaciones se ve que los equipos cuestionados poseen mayoritariamente valores mayores a 0,45, lo cual implícitamente se establece un criterio con respecto a la variable de los últimos valores de medida de cada equipo.

Figura 31: Factor de potencia 10kv y 2.5kv vs Comportamiento Historico



(a) Calificación del equipo vs factor depotencia 10kv

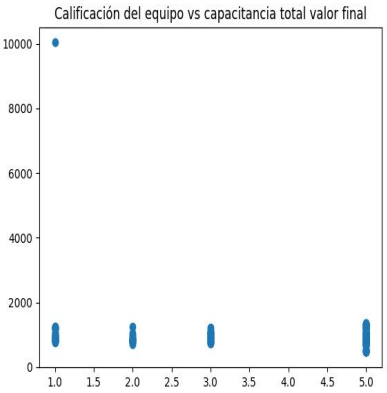


(b) Calificación del equipo vs Comportamiento historico

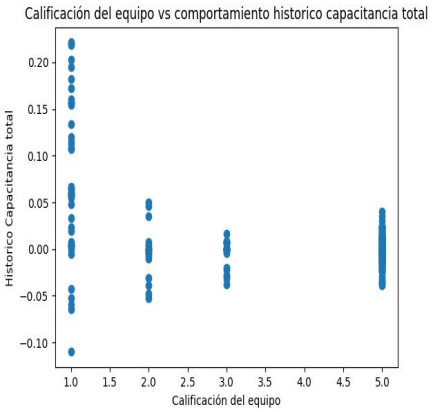
En los equipos que aparecen cuestionados y no poseen valores considerados "malos"(mayor a 0,45) seguramente son equipos cuestionados por otros aspectos y se puede notar en la **Figura 31.a** comparándola con la **Figura 31.b**que los equipos cuestionados poseen comportamientos normales (menores a 1,2) en sus valores históricos, y en sus últimos valores (menores a 0,45). Este análisis aplica para los valores de medida en 10kv y 2.5k, puesto que estos poseen normalmente valores muy similares. (arregla los pie de imagen de la figura 31)

Al ver que tanto para factor de potencia 2.5 como para 10, las variables comportamiento histórico y valor final se relacionan y a pesar de que el análisis de datos brinda ciertas luces acerca del comportamiento de los equipos en estas variables, los valores finales no poseen una regla explicita actualmente definida por los expertos, por lo que se decidió eliminar para el modelo las columnas que contienen los valores finales y dejar únicamente las que columnas que contienen el comportamiento histórico de estas medidas, logrando así la parsimonia para los modelos que se vayan a probar.

Figura 32: Capacitancia y su comportamiento historico vs Calificación



(a) Calificación del equipo vs ultimo valor



(b) Calificación del equipo vs comportamiento historico

- **Capacitancia**

  Para la capacitancia, podemos ver que para estos datos no hay mucha relación entre el ultimo valor medido con la variable calificación debido a que para todas las calificaciones se presentan valores similares en sus capacitancias como se observa en la figura **Figura 32.a** mientras que en la **Figura 32.b** la calificación con respecto al comportamiento histórico presenta poca uniformidad . Esto se puede explicar, teniendo en cuenta que los equipos varían en su funcionamiento debido a que no todos son fabricados por las mismas empresas.

  Así, podemos afirmar que los valores finales no poseen necesariamente relación con la calificación, puesto que hay equipos con valores cercanos a los 1500 y otros cercanos a 650 y no significa que están funcionando mal. Enfocándonos en la **Figura 32.b** , se puede ver que los equipos que son calificados en 5 concentran sus valores entre -0,04 y 0,04, que según al grupo que pertenezca el equipo, estos no son cuestionados. Para la calificación 1, el comportamiento histórico se acumula en valores mayores a 0.045, teniendo incluso valores mayores a 0.2, por lo que datos concuerdan con las consideraciones de los expertos. Similar al factor de potencia, los casos en el que los equipos son cuestionados pero que presentan valores normales en sus históricos, son equipos que seguramente son cuestionados por otras variables.

- **Medida de resistencia aislamiento**

  La figura **Figura 33** demuestra que los núcleos se comportan muy parecido, así que el análisis de un único núcleo se generaliza para todos los núcleos.

  Según los criterios definidos para las medidas de resistencia aislamiento, analizando la **Figura 33** se puede observar que bajo la escala de esta grafica no hay muchas ocurrencias donde los equipos sean cuestionados por algunos de estos núcleo por lo que es probable que sea necesario que se creen datos que representen todos los posibles casos en los que los núcleos cuestionen a los transformadores en conjunto con el resto de variables.

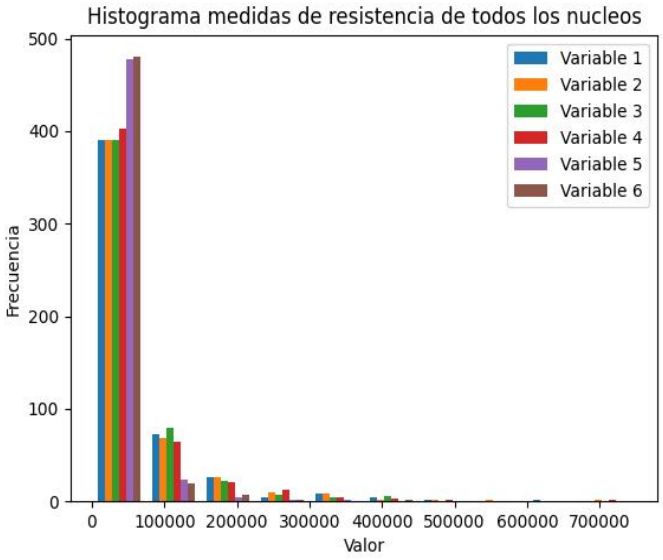


Figura 33: Histograma medidas de resistencia de todos los nucleos

3.4. Estudio general de modelos de clasificación con los datos de SAP

Debido a la naturaleza de los datos, las relaciones definidas por los expertos y las relaciones descritas a lo largo del análisis de estos datos, se puede afirmar con certeza que este problema en el entorno del aprendizaje automático pertenece al grupo de los problemas de clasificación.

Luego de analizar profundamente los datos extraídos de SAP, se proceder a indagar que modelo de Aprendizaje automático sería el adecuado. Dado que estos datos y las relaciones establecidas y encontradas encajan en la idea base de los algoritmos de clasificación, se enfocó la mirada en los árboles de decisión. Los árboles de decisión predicen para cada observación a que clase pertenece dado las observaciones de entrenamiento más frecuentes en la región a la que pertenece y esta descripción encaja perfectamente en la manera en la que se razona a la hora de cuestionar un equipo.

Además de este modelo se probarán bosques aleatorios, técnicas de boosting con el clasificador XGB y clasificador Lightgbm, aprovechando que estos modelos se basan en los árboles de decisión, pero usan técnicas que ayudan a ser más eficientes.

Como otro punto de estudio, se tiene el hecho de que los datos están claramente desbalanceados, por lo tanto, es posible que esto impacte de manera negativa a los modelos ya que no serán capaces de predecir correctamente a que clase pertenecen al no conocer todas las regiones de calificación de los criterios definidos por los expertos, así este estudio también estará enfocado en como los datos desbalanceados impactan en el rendimiento del modelo a la hora de calificar los equipos.

Se usará sklearn como el paquete para ejecutar los modelos de arboles de decisión y random forest.

3.4.1. Arboles de decisión

- **Entrenamiento del Modelo**

  Primero se mezclan los datos de manera aleatoria y se crean los datos de entrenamiento y de prueba

en una proporción 80 de entrenamiento 20 de prueba. Esto se hará con todos los modelos como se observa en la **Figura 34**

```
[9]
#Mezclar datos
df_sai_copy3=df_sai_copy3.sample(frac=1).reset_index(drop=True)

X = df_sai_copy3.drop(['calificacion_del_equipo'], axis=1)

#CARACTERÍSTICA OBJETIVO
y = df_sai_copy3['calificacion_del_equipo']

X = np.array(X) #Para arboles, esto no es necesario aun.
y = np.array(y)

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.20, #80% datos de entrenamiento, 20% prueba
                                                    stratify = y,
                                                    random_state = 1)
```

Figura 34: Mezcla de datos

Luego de esto se procede a instanciar un árbol con el fin de ajustarlo a nuestros datos. Podemos notar que para este algoritmo de árbol de decisión, están los parámetros min_samples_split , min_samples_leaf los cuales sirven para garantizar que múltiples muestras informan cada decisión del árbol, controlando qué divisiones se tendrán en cuenta. Un número muy pequeño suele significar que el árbol se ajustará en exceso, mientras que un número grande impedirá que el árbol aprenda los datos.

Otro parámetro importante es max depth que se utiliza para controlar el tamaño del árbol y evitar el sobreajuste. Se probaron múltiples valores y los resultados muestran que a valores altos en ambos parámetros los valores de la exactitud empeoran considerablemente mientras que a valores reducidos, la exactitud en los conjuntos de entrenamiento y de prueba mejoran considerablemente, tal y como se puede observar en la **Figura 35**, pero esto no significa que el modelo es el adecuado, se deben revisar otras métricas y realizar otro tipo de pruebas más específicas como pruebas unitarias. A continuación, se tienen además los resultados de la exactitud con ambas particiones de los datos

Figura 35: Ajuste de modelo de Arbol de decisión

- **Metricas de desempeño**

  Para las métricas de desempeño se usarán las medidas de precisión, recall y f1-score. La precisión y recall es una medida útil del éxito de la predicción cuando las clases están muy desequilibradas. En la recuperación de información, la precisión es una medida de la relevancia de los resultados, mientras que el recall es la fracción de instancias relevantes que se han obtenido sobre la cantidad total de instancias relevantes. Estas medidas se calculan de la siguiente manera :

  - **Precisión**

    $$P = \frac{T_p}{T_p + F_p}$$

    donde F_p son los falsos positivos y T_p son los positivos verdaderos.

  - **Recall**

    $$R = \frac{T_p}{T_p + F_n}$$

    donde F_n son los falsos negativos.

    Un sistema con un recall alto pero una precisión baja devuelve muchos resultados, pero la mayoría de sus etiquetas predichas son incorrectas cuando se comparan con las etiquetas de entrenamiento. Un sistema con alta precisión y baja recuperación es justo lo contrario:

devuelve muy pocos resultados, pero la mayoría de las etiquetas que predice son correctas en comparación con las etiquetas de entrenamiento. Un sistema ideal con alta precisión y recuperación devolverá muchos resultados, todos ellos etiquetados correctamente.

Estas cantidades se relacionan con el puntaje F_1, el cual está definido como la media harmonica de la precisión y el recall.

- F_1 **Score**

$$F1 = 2\frac{P \times R}{P + R}$$

Estas metricas se usarán para todos los modelos que se van a probar para este problema.

En la **Figura 36** se muestran las métricas del modelo sobre los datos de prueba. Se puede notar que las métricas para este modelo no arrojaron buenos resultados a pesar de que la exactitud obtenida fue cercana al 84 porciento. El recall indica que el modelo no captó toda la información suficiente de los datos para la calificación 2 y 3 y la precisión muestra que, para cada calificación, el porcentaje de calificaciones que fueron correctas fue muy baja, así que estas métricas son una prueba de que la exactitud no es una métrica fiable a la hora de decidir si un modelo es bueno o no.

```
decoded_y_test = y_test
decoded_predictions = decision_tree.predict(X_test)
print(f'Reporte de clasificación:')
print(classification_report(decoded_y_test,
                            decoded_predictions))
```

```
Reporte de clasificación:
              precision    recall  f1-score   support

           1       0.73      0.89      0.80         9
           2       0.20      0.25      0.22         4
           3       0.38      0.50      0.43         6
           5       0.95      0.89      0.92        83

    accuracy                           0.84       102
   macro avg       0.56      0.63      0.59       102
weighted avg       0.87      0.84      0.85       102
```

Figura 36: Metricas de arbol de decisión

- **Matriz de confusión**

  Con una matriz de confusión se pueden entender mejor las métricas anteriormente descritas. Una matriz de confusión es una tabla resumida que se utiliza para evaluar el rendimiento de un modelo de clasificación. Como podemos observar en la figura **Figura 37** En la diagonal de esta matriz se encuentran el número de predicciones correctas del modelo, por ejemplo, se calificaron adecuadamente 74 equipos en 5 mientras que hay 9 equipos que debieron ser calificados 5 fueron calificados 1 2 o 3.

  Para el caso de los equipos calificados 1, 8 equipos fueron bien calificados y solo en un caso el modelo cometió un error calificándolo 3, pero esto no puede pasar puesto que un equipo que realmente deba ser diagnosticado 1 es un equipo que puede presentar fallas funcionales que pueden derivar a accidentes. Gracias a la matriz de confusión se pueden estudiar más a fondo estos casos para así verificar la efectividad del modelo y poder seguir indagando de posibles caminos más adecuados para encontrar la solución

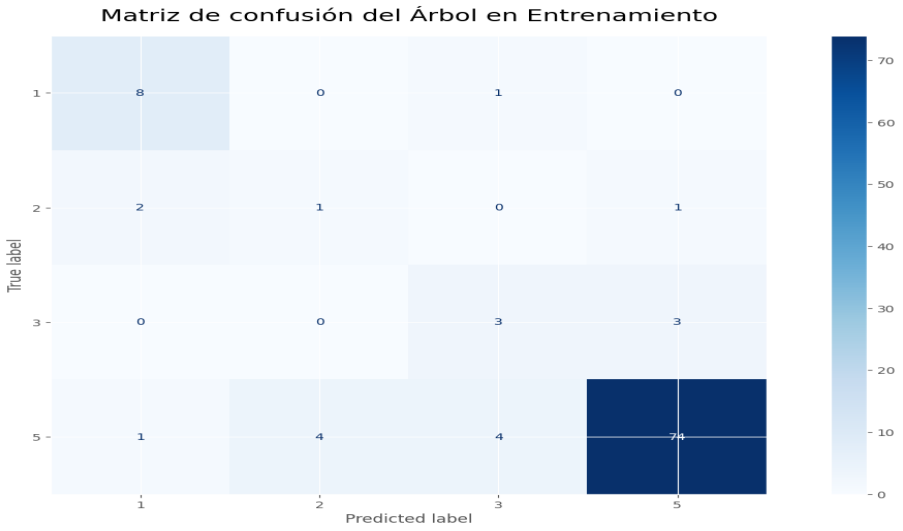


Figura 37: Matriz de confusión Arbol de decisión

- **Pruebas unitarias**

  Estas pruebas se realizan con el fin de probar casos específicos que muestren si el modelo es apto o no para ponerse en productivo. Se deben probar distintos casos con el fin de ver si el modelo fue capaz de entender la variedad de casos que poseen las calificaciones de los transformadores de corriente sin TAP pero como vemos en la **Figura 38**, con la supervisión del experto en transformadores de corriente de ISA, se corrobora que el modelo calificó mal, dando por sentado que el equipo está en buenas condiciones cuando los valores de capacitancia dado el fabricante están muy altos

Figura 38: Prueba unitaria árbol de decisión

Podemos concluir con estos resultados que los buenos números en algunas métricas no garantizan que el modelo sea el adecuado para solucionar algún problema, puesto que números muy altos pueden ser sinónimo de sobre ajuste además de que se tienen datos desbalanceados lo que hace que el árbol se incline hacia las clases dominantes, y en este caso las clases dominantes son los equipos calificados 5, por lo tanto la idea de la creación de datos sintéticos para el balanceo de clases toma más fuerza para tener un modelo más robusto y adecuado para ofrecer como una solución eficaz.

A continuación, se presentarán resultados en otras técnicas de árbol de decisión las cuales son bosques aleatorios que es un caso particular del bagging y gradient boosting el cual es implementado en el algoritmo XGBoost de Python. Podremos observar a lo largo de las imágenes que los resultados son similares al árbol de decisión en todas las métricas demostrando que los modelos están sobre ajustados.

3.4.2. Bosques aleatorios

- **Entrenamiento del Modelo** Se puede observar que los parámetros para los bosques aleatorios en Sklearn son similares a los de árboles de decisión. Se hicieron pruebas modificando los parámetros y se apreció sobreajuste en todas estas pruebas. En la **Figura 39**se puede observar que para los parámetros min_samples_split , min_samples_leaf se les asignaron valores más altos que para árbol de decisión y a diferencia de este, la métrica de exactitud se mantuvo alta tanto para los datos de entrenamiento como los de prueba.

```
[52]
from sklearn.ensemble import RandomForestClassifier


decision_tree_f = RandomForestClassifier(criterion='gini',
                                         min_samples_split=10, #Muestras minimas por nodo
                                         min_samples_leaf=9, #Muestras minimas por hoja
                                         max_depth = 21,
                                         random_state=2)

# Ajustamos el modelo con los datos
decision_tree_f.fit(X_train, y_train)

#Exactitud
y_pred_train = decision_tree_f.predict(X_train)

#Porcentaje de precision con los datos de entrenamiento
acc_train = accuracy_score(y_train, y_pred_train)
print("Exactitud con datos de entrenamiento: {:.2f}".format(acc_train))

y_pred_test = decision_tree_f.predict(X_test)

#Porcentaje de precision con pruebas
acc_test = accuracy_score(y_test, y_pred_test)
print("Exactitud con datos de pruebas: {:.2f}".format(acc_test))
```

```
Exactitud con datos de entrenamiento: 0.89
Exactitud con datos de pruebas: 0.87
```

Figura 39: Entrenamiento random forest

- **Metricas modelo Random Forest**

  A pesar de que para la calificación 1 y 5 se obtuvo una mejor puntuación en su precisión y recall, en las otras calificaciones no obtuvo buen desempeño y esto es debido que los datos están desbalanceados y el modelo no obtuvo suficiente información para ser capaz calificar equipos en 2 o en 3.

```
decoded_y_test = y_test
decoded_predictions = decision_tree_f.predict(X_test)
print(f'Reporte de clasificación:')
print(classification_report(decoded_y_test,
                            decoded_predictions))
```

```
Reporte de clasificación:
              precision    recall  f1-score   support

           1       1.00      0.67      0.80         9
           2       0.00      0.00      0.00         4
           3       0.00      0.00      0.00         6
           5       0.86      1.00      0.93        83

    accuracy                           0.87       102
   macro avg       0.47      0.42      0.43       102
weighted avg       0.79      0.87      0.83       102
```

Figura 40: Metricas Random Forest

- **Matriz de confusión** En esta matriz de confusión se puede verificar con mayor claridad las metricas de precisión y recall. En este caso, el modelo calificó todos los equipos que debían ser calificados con 2 como un 5, lo cual es un fallo muy grave bajo los estándares de nuestro problema. Similarmente, todos los equipos que debieron ser calificados 3 fueron calificados 5. Tener los datos tan desbalanceados hace que el modelo tenga una clase dominante y no sea capaz de generar regiones adecuadas para las otras calificaciones.

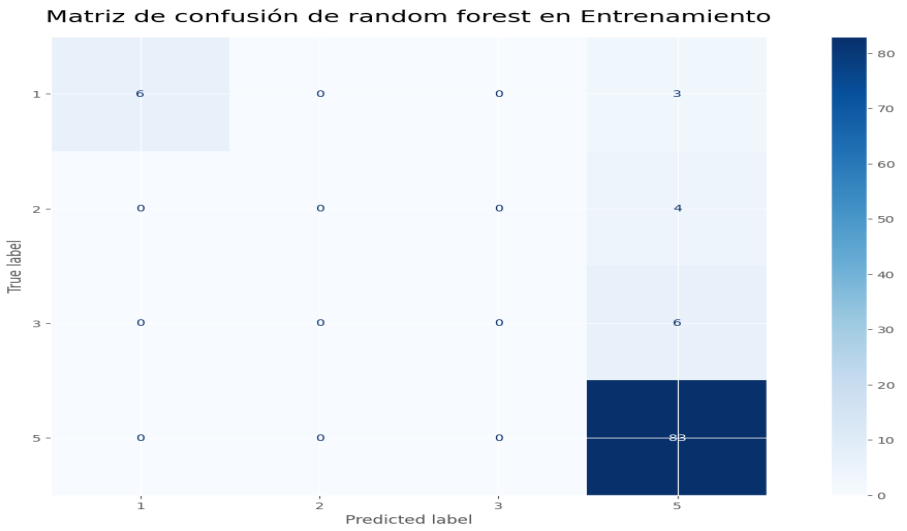


Figura 41: Matriz de confusión

- **Pruebas unitarias**

  Las pruebas unitarias corroboran lo concluido gracias a las otras métricas, el modelo no es capaz de calificar en clases que no sean 1 o 5. En la figura 40 se observa un equipo que debe ser calificado 3 pero es calificado 1

```
#Prueba unitaria
#VARIABLES
#capaciancia,factor10,factor2.5,capacitancia_ultimo,factor_ultimo,factor25_ultimo,nucle1,ncle2,nucle3,nucle4
#,nucle5,nucle6,severidad,familia,cajetin,porcelana,fuga,diafrag,insp_visual,nivel_aceite
X_test_c1=np.array(X_test)

unit_test=np.array([[0.03,1,1,1224,0.6, 0.18,401,1163,1934,1236,2150,0,5,3,5,5,5,5,5,5],
                    [0.008,1.18,1.08151,750,0.112154,0.25815151,5000,100000,550,205,-1,-1,5,3,5,5,5,5,3,5]])
#print(X_test_c1.shape,unit_test.shape)
unit_test1=np.concatenate((X_test_c1,unit_test),axis=0)
p=decision_tree_f.predict(unit_test1)
prob=decision_tree_f.predict_proba(unit_test1)
print(f"Label: "+ str(p[len(unit_test1)-2]) + ", Con una confianza de "+ str(np.max(prob[len(unit_test1)-2])*

Label: 1, Con una confianza de 34.066305470281236%.
```

Figura 42: Pruebas unitarias

3.4.3. Clasificador XGBoost

Para este modelo se entrará más en detalle puesto que es el modelo más robusto y posee parámetros diferentes. XGBoost es una librería distribuida optimizada de gradient boosting diseñada para ser altamente eficiente, flexible y portable. Implementa algoritmos de aprendizaje automático bajo el marco Gradient Boosting. Los parámetros más relevantes para el problema son :

- **booster** Este parámetro se deja por defecto como gbtree puesto que estamos trabajando con un

modelo basado en árbol.

- **max_depth**: Este parámetro es usado para controlar la máxima profundidad de los árboles. El valor de este parámetro depende del problema y será buscado con grid search.
- **max_leaf_nodes**: Es el número máximo de nodos terminales u hojas de un árbol. Se puede definir en lugar de max_depth. Dado que se crean árboles binarios, una profundidad de 'n' produciría un máximo de 2n hojas.
- **objective**: Esto define la función de pérdida que debe minimizarse. La elección de esta función de perdida depende de la naturaleza de la variable respuesta, en este caso es un problema de clasificación de 4 clases que serían en este caso las calificaciones, así que se usa el parámetro 'multi:sofprob' que nos indica la clase a la que pertenece la el equipo en cuestión y devuelve la probabilidad prevista de que cada punto de datos pertenezca a cada clase.
- **n_estimators**: Es el número de árboles en que serán usados para el algoritmo de boosting.
- **learning_rate**:Hace que el modelo sea más robusto reduciendo los pesos en cada paso.

  El resto de los parámetros se dejan por defecto y se estudiaran más a fondo en posteriores pruebas.

- **Entrenamiento del Modelo**

  Se ajustó un modelo con valores por defecto para ver qué tipo de resultados se obtienen con este tipo algoritmos robustos. Como se puede observar, la exactitud con respecto a los otros modelos mejoró considerablemente teniendo en cuenta que no se ajustó el modelo con los mejores parámetros posibles para estos datos. Sin embargo, dado que los datos están desbalanceados no se puede afirmar con total certeza que el modelo es capaz de calificar correctamente los equipos.

```
from xgboost import XGBClassifier
xgbc=XGBClassifier()
XGBClassifier(base_score=0.5,
              booster='gbtree',
              colsample_bylevel=1,
              colsample_bynode=1,
              colsample_bytree=1,
              gamma=0,
              learning_rate=0.01,
              max_delta_step=0,
              max_depth=21,
              min_child_weight=1,
              missing=1,
              n_estimators=200,
              n_jobs=1,
              nthread=None,
              objective='multi:softprob',
              random_state=2,
              reg_alpha=0,
              reg_lambda=1,
              scale_pos_weight=1,
              seed=None,
              silent=None,
              subsample=1,
              verbosity=1)
xgbc.fit(X_train, y_train)
y_pred = xgbc.predict(X_test)
print('Exactitud de los datos de entrenamiento: {:.2f}'
      .format(xgbc.score(X_train, y_train)))
print('Exactitud de los datos de prueba {:.2f}'
      .format(xgbc.score(X_test, y_test)))
```

```
Exactitud de los datos de entrenamiento: 1.00
Exactitud de los datos de prueba 0.96
```

Figura 43: Ajuste de modelo XGBoost

- **Metricas de desempeño**

Reporte de clasificación:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2
1	1.00	1.00	1.00	1
2	1.00	0.50	0.67	2
3	0.95	1.00	0.98	21
accuracy			0.96	26
macro avg	0.99	0.88	0.91	26
weighted avg	0.96	0.96	0.96	26

Figura 44: Metricas de desempeño XGBoost

- **Matriz de confusión**

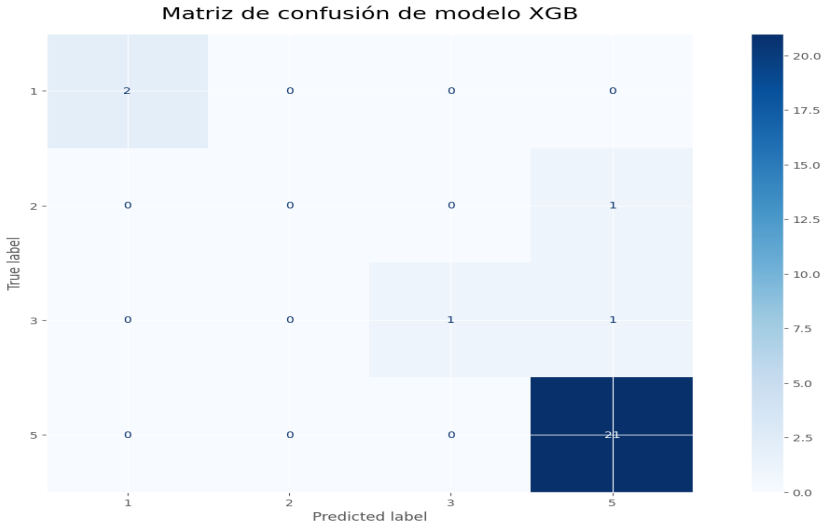


Figura 45: Matriz de confusión

- **Pruebas unitarias**

```
#Prueba unitaria
#VARIABLES
#capaciancia,factor10,factor2.5,capacitancia_ultimo,factor_ultimo,factor25_ultimo,nucle1,ncle2,nucle3,nucle4
#,nucle5,nucle6,severidad,familia,cajetin,porcelana,fuga,diafrag,insp_visual,nivel_aceite
X_test_c1=np.array(X_test)

unit_test=np.array([[0.02,1,1,1300,0.4,0.2,500,500,500,500,500,500,5,3,5,5,5,5,5,5],
                    [0.008,1.18,1.08151,750,0.112154,0.25815151,5000,100000,550,205,-1,-1,5,3,5,5,5,5,3,5]])
#print(X_test_c1.shape,unit_test.shape)
unit_test1=np.concatenate((X_test_c1,unit_test),axis=0)
p=xgbc.predict(unit_test1)
prob=xgbc.predict_proba(unit_test1)
print(f"Label: "+ str(p[len(unit_test1)-2]) + ", Con una confianza de "+ str(np.max(prob[len(unit_test1)-2])*1
print("Segundo Label mas confiable: " + str(list(prob[len(unit_test1)-2]).index(sorted(prob[len(unit_test1)-2])

Label: 2, Con una confianza de 94.393861293792725.
Segundo Label mas confiable: 0 Con una confianza del 4.6604715287685395
```

Figura 46: Prueba unitaria

- **importancia de datos**

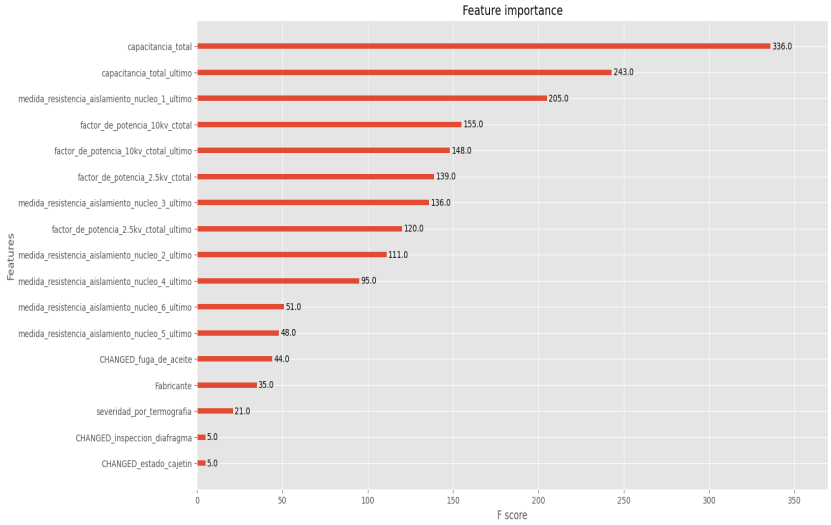


Figura 47: Importancia de datos

Los resultados obtenidos dan entender que cualquiera de los tres modelos es adecuado para el tipo de problema que se está trabajando, puesto que es capaz de predecir calificaciones de los equipos dada ciertas observaciones. Aunque las métricas que muestran los modelos son números prometedores, al estar los datos muy desbalanceados, las pruebas unitarias realizadas tienden a arrojar resultados erróneos ya que no son capaces de generar una región adecuada para todas las posibles observaciones, por lo tanto, estos modelos confunden las calificaciones y no son lo suficientemente confiables para calificar estos equipos.

4. Creación de datos sinteticos

Como medida remedial para el desbalanceo de clases, se tomó la decisión de desarrollar una metodología para la creación de datos basados en las múltiples reglas de diagnóstico de todas las variables. La base fundamental de la creación de estos datos es el método de la transformada inversa el cual nos permite construir cualquier distribución a partir de una distribución uniforme (0,1)

La creación de datos debe ser muy minuciosa y fiel a las reglas y posibles situaciones en un entorno productivo por lo que se siguieron las siguientes pautas:

- Las reglas principales para cualquier equipo son que, si al menos una de las variables cuestiona, el equipo puede ser calificado y si el equipo posee múltiples variables cuestionando, la calificación la determina la variable que califique más drásticamente, es decir, si hay dos variables donde una califica 1 y otra 2, la variable que decidirá la calificación será la que califica 1.
- La variable **Medida de resistencia aislamiento** ddepende de la existencia de los núcleos y esto depende del fabricante del transformador. Debido a que esta variable es la que más posee casos para crear, la creación de datos se tuvo que sostener con base a estas variables que, según las pruebas, son las que más influyen en el aprendizaje del modelo.
- Los casos que se obtengan de las variables categóricas deben acompañar todos los casos de las demás variables con el fin de evitar sesgos en las calificaciones.
- Se debió verificar constantemente las calificaciones de las observaciones creadas para evitar contradicciones en nuestros datos las cuales causan que el modelo realice calificaciones erróneas

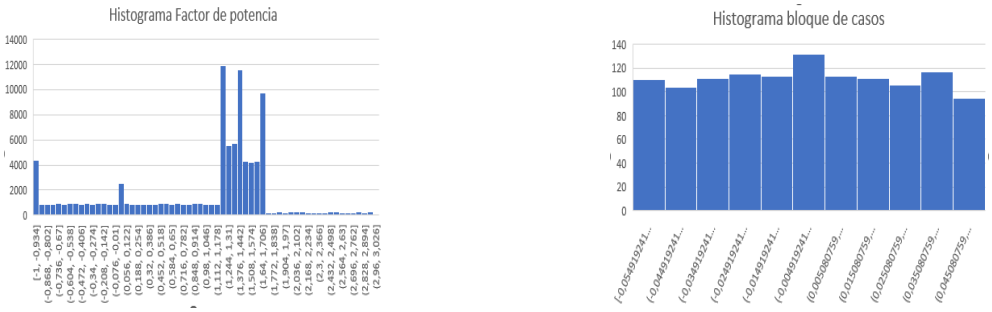
Teniendo en cuenta la estructura de las reglas de diagnóstico para las variables cuantitativas, al ser estos intervalos de la forma $a \leq x \leq b$, se debe garantizar la ocurrencia de una cantidad significativa de valores que pertenecen a ese intervalo, es decir, si tenemos un intervalo $51 \leq x < 201$ con $x \in Z$, con el método de la transformada inversa se crea una distribución de tal manera que se garantice que los subintervalos de la forma $51 \leq x < 100$ y $100 \leq x < 201$ posean la misma cantidad de datos y además que al generar los datos de manera aleatoria se tenga la certeza que la gran mayoría de valores en esos subintervalos están cubiertos

En la practica la segmentación de los subintervalos se realizó de la siguiente manera:

- Se tomó un porcentaje de entre 10 y 15 por ciento según el caso para los valores extremos de los intervalos debido a que esto ayuda al modelo a generar regiones de predicción más precisas.
- Teniendo la seguridad de la presencia de valores extremos se le asigna el resto de porcentaje de datos a los valores que están entre los extremos.
- Con el fin de asegurarse de que la gran mayoría de valores que se encuentran entre esos intervalos de cuestionamiento aparezcan en los datos, se crearon por caso bloques de 3726 datos que fue el número elegido luego de múltiples pruebas al impacto de los datos en el modelo.
- Este proceso se realizó para cada caso entre las posibles combinaciones de calificación de los equipos.

La **figura 48** muestra el histograma de unos de los múltiples subintervalos creados para cada región de cuestionamiento. Se puede observar que cada subintervalo posee cantidades de datos similares por lo que la generación de grandes cantidades de datos ayudó a que tener uniformidad en la cantidad de

Figura 48: Aplicación metodo Tranformada inversa



casos y evitar el fenómeno de desbalanceo de clases, el cual impacta negativamente en el entrenamiento de modelos basados en arboles de decisión. Con esta metodología, se logró obtener datos saludables y no redundantes con el fin de que el modelo aprenda todas las reglas sin espacio a la confusión. En total fueron necesarios aproximadamente 3 millones de datos, pero que garantizaran tener regiones de predicción más adecuadas a nuestras reglas

5. Resultados finales

Teniendo ya los datos consolidados, se probarán dos modelos los cuales están basados en la técnica Gradient Boosting, las cuales son XGBoost y LightGBM

5.1. XGBOOST

Para este modelo se usó Grid search Cv el cual usa Validación cruzada (CV) para encontrar el modelo que mejor se ajuste a nuestros datos dado una malla de hiperparámetros. Luego de muchas pruebas se determinó que eran los hiperparámetros que más influían en el entrenamiento del modelo son los siguientes:

- **Learning Rate** : La reducción del tamaño del paso se utiliza en la actualización para evitar el sobreajuste. Después de cada paso de refuerzo, podemos obtener directamente los pesos de las nuevas características, y el hiperparámetro learning rate reduce los pesos de las características para que el proceso de refuerzo sea más conservador.
- **Max depth** : Profundidad máxima de un árbol. Aumentar este valor hará que el modelo sea más complejo y propenso a sobre ajustarse
- **Number of estimators**:Este hiperparámetro controla la cantidad de árboles que sean agrupados y entrenados.

Los resultados obtenidos son los de la **Figura 49**

```
{'learning_rate': 0.4, 'max_depth': 21, 'n_estimators': 50}
```

Figura 49: hiperparámetros encontrados con grid search

Ya obtenidos estos parámetros se procede a instanciar el modelo para entrenarlo con la siguiente configuración:

```
from xgboost import XGBClassifier

#xgbc_grid=XGBClassifier()
xgbc_prueba=XGBClassifier(base_score=0.5,max_depth=21,
              booster='gbtree',
              learning_rate=0.4,
              missing=1,
              n_estimators=50,
              nthread=None,
              min_split_loss=1,
              objective='multi:softprob',
              min_child_weight=0,
              subsample=0.4,
              colsample_bylevel=0.5,
              colsample_bynode=1,
              max_cat_threshold=5,
              verbosity=1,eval_metric='auc')
```

Figura 50: hiperparámetros encontrados con grid search

El resto de hiperparámetros fueron determinados a partir de exhaustivas pruebas y se concluyó que la gran mayoría de hiperparámetros funcionan bien con sus valores por defecto.

Ya entrenado este modelo se obtuvieron las siguientes métricas:

- **Precisión** ,**Recall** y **F- score**

Reporte de clasificación:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	128
1	0.99	1.00	1.00	135
2	1.00	0.99	1.00	114
3	1.00	1.00	1.00	90
accuracy			1.00	467
macro avg	1.00	1.00	1.00	467
weighted avg	1.00	1.00	1.00	467

Figura 51: Recall, precisión y F-score modelo XGBoost

En la **Figura 51** podemos observar las métricas principales de este tipo de modelos. En casi todas las clases la precisión fue del 100 por ciento exceptuando la clase 1 el cual tuvo aparentemente un error así que se debió investigar cual fue la observación que hizo que el modelo se equivocara

CALIFICACION_VERDADERA	PREDICCION	%CONFIANZA	VERIFICACIÓN	CONTEO DE ERRORES
1	1	100	0	0
2	2	99,00000095	0	
3	3	100	0	
1	1	100	0	
2	2	100	0	
3	3	100	0	
0	0	100	0	
1	1	100	0	
0	0	100	0	

Figura 52: Validación de las métricas

Se puede observar que el modelo no cometió un error y fue un simple error del algoritmo encargado de mostrar los valores de las métricas. Para el recall, se puede observar una situación muy similar a la precisión y es que en la mayoría de los casos el modelo obtuvo el 100 por ciento de la información de los datos excepto la clase 2 la cual tuvo un recall de 0.99.

Luego de múltiples pruebas con las pruebas unitarias no se logró encontrar alguna observación que permitiera ver donde el modelo puede estar fallando, por lo que posiblemente es un simple error del algoritmo al momento de mostrar los resultados

Por último tenemos el F-score que nos permite analizar las ambas métricas al tiempo. En este caso, el F-score de este modelo obtuvo el 100 por ciento de la calificación, por lo tanto, nuestros datos de prueba indican que el modelo aprendió muy bien y es capaz de calificar una gran cantidad de observaciones sin equivocarse o cometer errores graves como no cuestionar un equipo o ?sub-cuestionarlo?, es decir, darle una calificación más flexible y omitir posibles daños en los transformadores.

- **Validación cruzada**

```
scores = cross_val_score(xgbc_prueba, df_x_train_c2, y_train_c2, cv=3,n_jobs=5)
print("Mean cross-validation score: %.2f" % scores.mean())

kfold = KFold(n_splits=10, shuffle=True)
kf_cv_scores = cross_val_score(xgbc_prueba, df_x_train_c2, y_train_c2, cv=kfold,n_jobs=5)
```

```
Mean cross-validation score: 0.99
```

Figura 53: validacion Cruzada XGBoost

- **Importancia de variables**

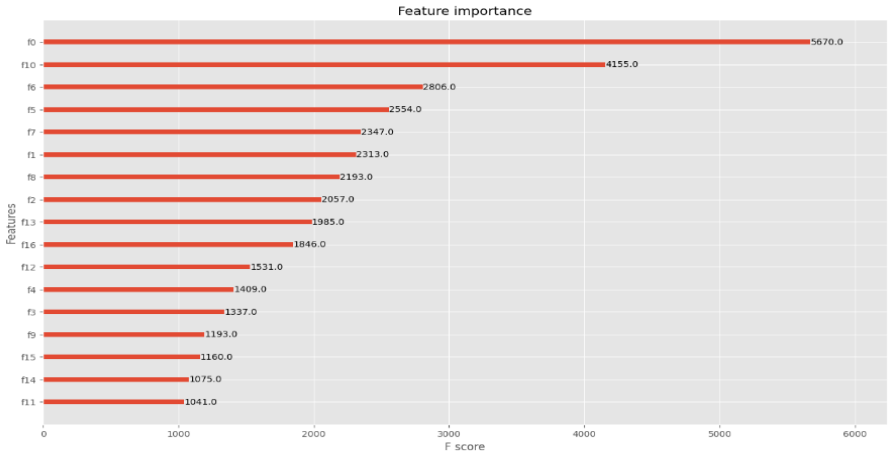


Figura 54: Importancia de variables por impureza de nodo

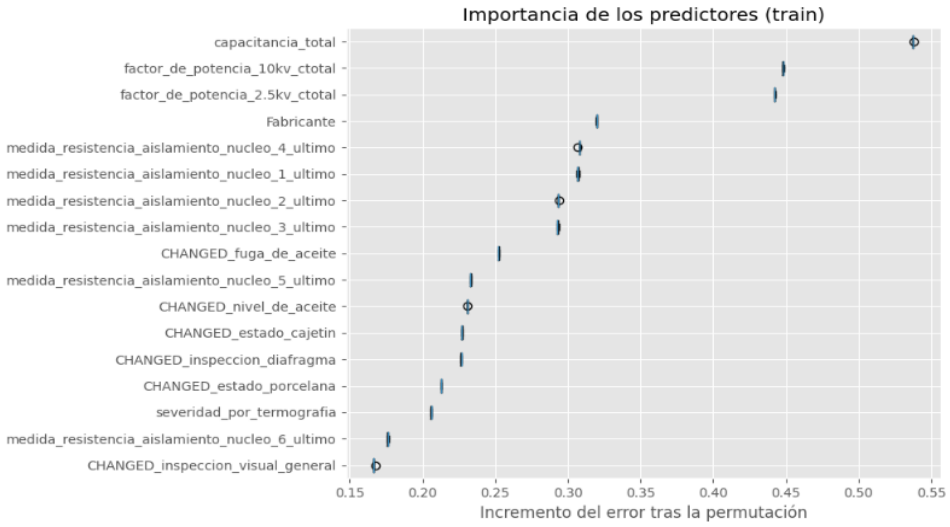


Figura 55: Importancia de variables por permutación

Se puede observar en la **Figura 53** que bajo el criterio de importancia de variables por impureza de nodos, la variable más importante es la capacitancia, lo cual es normal debido a se crearon muchos datos para esta variable debido a su importancia y dificultad a la hora de implantar la reglas en el modelo.

La siguiente variable más importante con respecto a las demás es la variable **Fabricante** y este es un aspecto muy positivo puesto que las reglas para las variables **Capacitancia** y **Factor de potencia** cambian según el grupo o fabricantes al que pertenecen y es necesario que el modelo tenga presente esa diferenciación.

Para el resto de las variables la importancia es equilibrada en ciertos grupos de variables por lo que el modelo es capaz de asignarle el peso adecuado a cada variable según la decisión que se tome.

Para la **figura 54** que representa el criterio de importancia por permutación, la situación no es muy diferente; La variable capacitancia sigue siendo la variable más importante con diferencia al resto, pero se puede observar que la variable fabricante perdió importancia.

Desde el punto de vista de la importancia por permutación, esto quiere decir que al realizar permutaciones con los datos la variable fabricante no impactó mucho en el incremento del error del modelo por lo que el fabricante no influye tanto como las variables capacitancia y factor de potencia, pero se sigue manteniendo como una de las variables más importantes para el modelo lo cual es positivo al momento de realizar predicciones. Para el resto de las variables, la importancia con respecto a la permutación no varió mucho en comparación con la importancia de variables con respecto a la impureza de nodos.

- **Curva ROC**

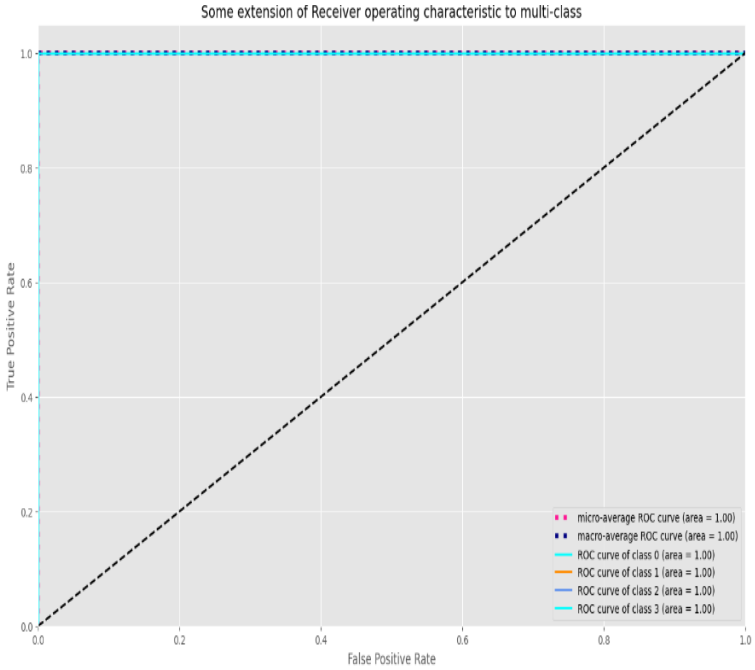


Figura 56: Curva ROC modelo XGBC

La curva ROC para este modelo mostrada en la figura x indica que para cada clase o calificación, todas las predicciones son verdaderos positivos, por lo tanto la curva asignada a cada calificación bordea todo el rincón superior izquierdo del grafico lo cuales un indicio de alta sensibilidad, es decir todos los casos positivos dispuestos en los datos se clasifican correctamente por lo que el área bajo la curva ROC en cada clase es 1, mostrando que nuestro modelo califica adecuadamente todas las observaciones desconocidas que se usaron para la validación

5.2. LigthGBM

```
LGBMClassifier
LGBMClassifier(boosting='gbdt', device_type='gpu', learning_rate=0.05,
               max_depth=35, metric='multi_logloss', n_estimators=95,
               num_class=5, num_leaves=90, objective='multiclass',
               tree_learner='feature_parallel')
```

Figura 57: hiperparámetros modelo gradient boosting lightGBM

- **Precisión, Recall y F- score**

```
Reporte de clasificación:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        64
           1       1.00      1.00      1.00        66
           2       1.00      1.00      1.00        59
           3       1.00      1.00      1.00        45

    accuracy                           1.00       234
   macro avg       1.00      1.00      1.00       234
weighted avg       1.00      1.00      1.00       234
```

Figura 58: hiperparámetros modelo gradient boosting lightGBM

- **Matriz de confusión**

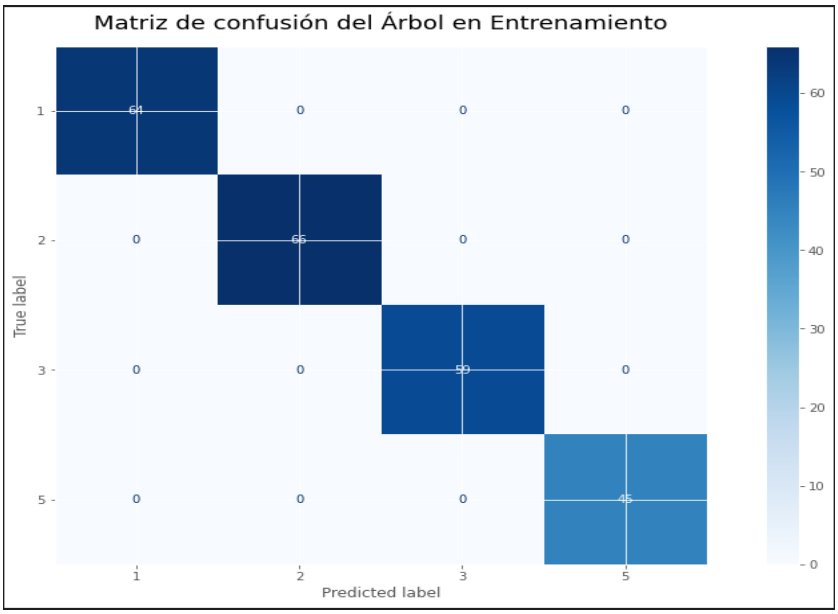


Figura 59: Matriz de confusión lightGBM

- **Validación cruzada**

```
scores = cross_val_score(clf, df_x_train_c2, y_train_c2, cv=3,n_jobs=5)
print("Mean cross-validation score: %.2f" % scores.mean())

kfold = KFold(n_splits=10, shuffle=True)
kf_cv_scores = cross_val_score(clf, df_x_train_c2, y_train_c2, cv=kfold,n_jobs=5)
```

✓ 4m 7.7s

```
Mean cross-validation score: 0.98
```

Figura 60: Validación cruzada lightGBM

- **Importancia de variables**

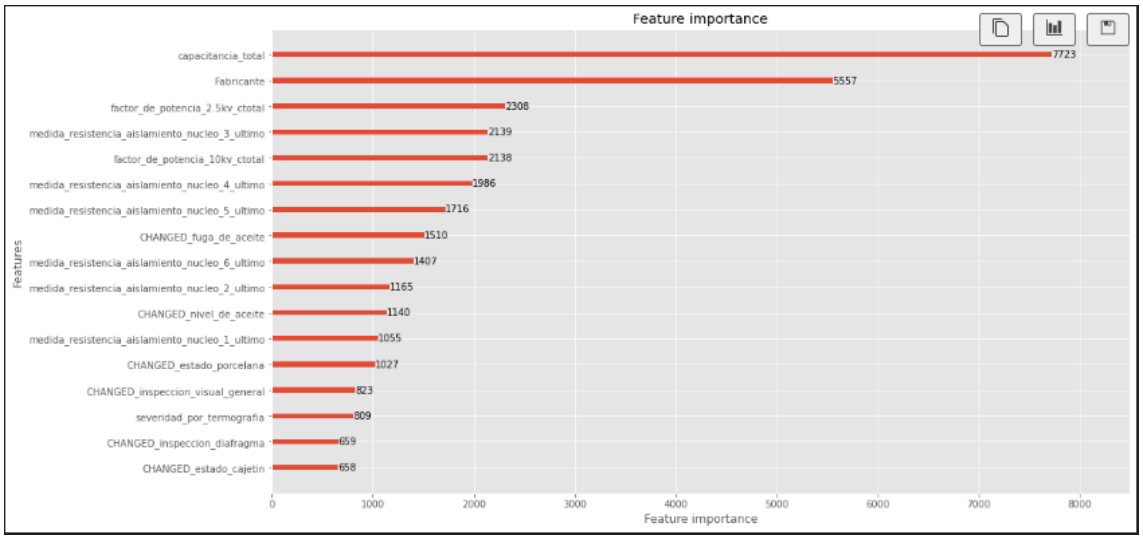


Figura 61: Importancia por impureza de nodos

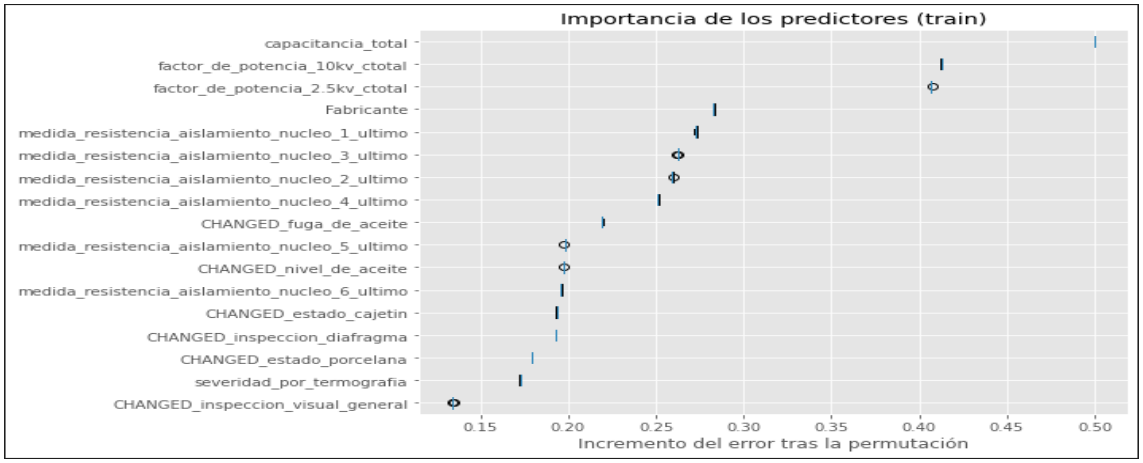


Figura 62: Importancia por permutación

Para el modelo desarrollado por Microsoft se necesitó usar más hiperparametros debido a que a pesar de que ambos modelos poseen en su centro el algoritmo de Gradient Boosting, la arquitectura de este algoritmo es diferente con respecto al anterior modelo por lo que el impacto de los hiperparametros varia muchísimo. En este caso, no se usó grid search debido al inmenso costo computacional debido a que hay muchos hiperparametros que analizar, pero se logró encontrar a partir de experimentación buenos hiperparametros que permitieron obtener muy buenos resultados.

Para este obtuvimos métricas muy similares al modelo desarrollado por XGB, además de que las pruebas unitarias realizadas con los expertos lanzaron resultados satisfactorios, por lo que se debe tener en cuenta otro criterio extra elegir el modelo mas adecuado para este problema, en este caso, se medirá la capacidad predictiva.

5.3. Capacidad predictiva

Luego de analizar ambos modelos, se puede observar en la **figura 61** que ambos son muy buenos calificando estos equipos pero en este caso XGBoost es el modelo que posee mayor capacidad predictiva , puesto que es el modelo que más confía en sus predicciones, y esto seguramente se debe a que se definieron bien las regiones de predicción del modelo y aprendió más de los datos que el modelo lightGBM y esto juega un papel importante en el entorno productivo puesto que el modelo deber ser confiable para los estándares de mantenimiento de la empresa.

Figura 63: Capacidad predictiva

CALIFICACION_VERDADERA	PREDICCION	%CONFIANZA
2.0	2	94.999999
2.0	2	95.999998
1.0	1	97.000003
2.0	2	97.000003
1.0	1	97.000003
...	...	...
0.0	0	100.000000
1.0	1	100.000000
3.0	3	100.000000
1.0	1	100.000000
2.0	2	100.000000

(a) XGBoost

CALIFICACION_VERDADERA	PREDICCION	%CONFIANZA
1.0	1	61.0
1.0	1	61.0
0.0	0	67.0
2.0	2	80.0
2.0	2	82.0
...	...	...
0.0	0	100.0
0.0	0	100.0
0.0	0	100.0
0.0	0	100.0
0.0	0	100.0

(b) LightGBM

6. Conclusión

- Uno de los modelos más adecuados para calificar equipos que poseen reglas complejas son los basados en gradient boosting.
- Para poder cubrir todos los casos de manera adecuada se deben diferenciar todas las posibles situaciones en las labores de mantenimiento según los valores de cada regla.
- Se necesitan grandes volúmenes de datos para poder tener un modelo adecuado para los estándares de mantenimiento.
- Estos modelos son capaces de aprender cualquier patrón siempre y cuando se creen datos adecuados
- Las métricas que miden la precisión y la capacidad predictiva de estos modelos pueden ser un buen indicio de cómo está nuestro modelo, pero no deben ser las ultimas conclusiones, las pruebas unitarias ayudan mucho a determinar si las regiones están bien definidas
- Estos modelos son capaces de calificar grandes volúmenes de equipos en tiempo real, por lo aumentaría la eficiencia de las labores de mantenimiento y disminuiría el costo computacional con respecto a algoritmos con lógica booleana o difusa.
- Los modelos de machine learning son muy portables y además, pueden adaptarse automáticamente a los cambios en los datos.
- La salud de los datos es vital para que los modelos aprendan bien, por lo que lo más adecuado es validar los datos de entrenamiento permanentemente con el fin de obtener metricas fieles a la realidad del modelo.

Referencias

- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2014). An Introduction to Statistical Learning: with Applications in R. Springer.
- Hastie, T., Tibshirani, R., & Friedman, J. H. (2013). The Elements of Statistical Learning: Data Mining, Inference, and Prediction
- Ross, S. M. (2012). Simulation. Academic Press.
- Gradient Boosting con python. (n.d.).https://www.cienciadedatos.net/documentos/py09_gradient_boosting_python.html
- XGBoost Documentation xgboost 1.7.5 documentation. (n.d.). https://xgboost.readthedocs.io/en/stable/index.html
- Wikipedia. (2022). Curva ROC. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Curva_ROC