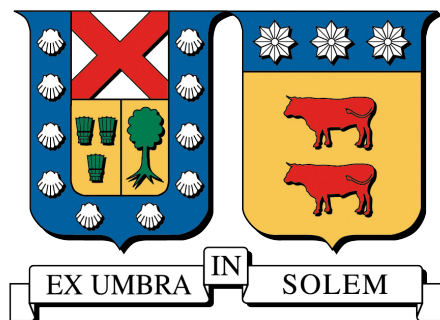


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INGENIERÍA MECÁNICA
VALPARAÍSO - CHILE



**“MEJORA AL CICLO DE MANTENIMIENTO DE LÍNEA DE FILTRADO
LAROX (CODELCO DIVISIÓN ANDINA) MEDIANTE LAS TÉCNICAS DE
MANTENIMIENTO CENTRADO EN CONFIABILIDAD Y PROCESAMIENTO
DEL LENGUAJE NATURAL”**

SEBASTIAN ANTONIO RIVAS MARCHANT

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MECÁNICO

PROFESOR GUÍA : LUIS PÉREZ POZO
PROFESOR CORREFERENTE : RAFAEL MENA YANSSEN

OCTUBRE 2022

AGRADECIMIENTOS

Quiero dedicar este trabajo a toda la gente que me acompañó durante este período de estudio.

A mi mamá Fresia, que sin su crianza y enseñanzas no podría haber logrado nada de lo que hoy tengo. A mi papá Eliecer, que me heredó su gusto por aprender, por las matemáticas, la mecánica y siempre me ayudó con mis materias, hasta el último año. A mis hermanas que me acompañaron durante todo este proceso. A mi polola Natalia, que ha sido mi cable a tierra, apoyándome incondicionalmente y ayudándome a centrarme en lo que es realmente importante para el futuro. Al profesor Luis, que tuvo siempre la mejor disposición para apoyarme en este trabajo, independiente de que haya tomado un tema que no es del área en que investiga principalmente. Y a todos mis amigos y cercanos que estuvieron conmigo a lo largo de mi carrera.

Muchas gracias.

Veni Vidi Vici

RESUMEN

La Corporación Nacional del Cobre de Chile (Codelco Chile) es una empresa estatal dedicada a la explotación minera. Dentro de sus divisiones a lo largo del país se encuentra la División Andina, ubicada en la V región, comuna de Los Andes. En esta división se cuenta con la Planta de Molibdeno y Filtrado de Cobre, donde se realiza la tarea de obtención y procesamiento del molibdeno para su venta, y de preparar el concentrado de cobre para ser dirigido a su refinación en la División Ventanas. En el proceso de filtrado de concentrados de cobre ocurre un número significativo de detenciones no programadas, por lo cual se está trabajando en mejorar el actual ciclo de mantenimiento mediante la predicción de fallas, para así evitar pérdidas en la productividad.

Como mejora al ciclo de mantenimiento en el proceso de filtrado Larox, es que en primer lugar se hace una revisión de las fallas reportadas durante el período de enero – junio 2021, para realizar un estudio de mantenimiento centrado en confiabilidad (RCM), el cual se utiliza como para los pasos posteriores del proyecto. Posteriormente, mediante el Sistema Operativo de Mantenimiento (SOMA) de Codelco y su matriz de criticidad, se evalúa el impacto de cada una de las fallas incluidas en el RCM, con el fin de verificar cuales son más críticas a nivel económico, ambiental, operacional y de seguridad.

En el proceso de filtrado Larox se cuenta con un sistema de reporte de avisos en SAP, en el cual los operadores, mantenedores e ingenieros reportan anomalías detectadas durante el funcionamiento de los equipos. Estos avisos se descargan en formato Excel, los cuales se catalogan con un nivel de importancia bajo, medio o alto dependiendo de lo obtenido en el estudio de criticidad realizado anteriormente. Una vez categorizados los avisos disponibles (420 avisos aproximadamente), se programa en Python un algoritmo de Procesamiento del Lenguaje Natural, un área de la Inteligencia Artificial encargada de comprender la forma en que se comunican los humanos, para así, luego de entrenar el algoritmo con la base de datos inicial, categorizar nuevos avisos ingresados. En este algoritmo en primer lugar se realiza una limpieza a la base de datos, para posteriormente ser pre-procesada y probar distintos métodos de clasificación, buscando el que obtenga un mayor porcentaje de precisión.

Una vez se desarrolla el algoritmo con mayor precisión, este se utiliza para programar un nuevo código en Python, el cual recibe una nueva base de datos descargada desde SAP, procesa los nuevos avisos cargados y entrega una tabla con el respectivo nivel de importancia predicho para cada uno de ellos. El algoritmo final posee una precisión de aproximadamente un 80 %, el cual es considerado significativamente favorable para futuras predicciones.

ABSTRACT

Codelco (Corporación Nacional del Cobre de Chile – National Copper Corporation of Chile) is a state company dedicated to mining. Along the country it has different branches, for example División Andina which is in the V region, specifically in Los Andes. In this branch, it can be found the Molybdenum and Copper Filtration Plant, where the molybdenum is processed for its posterior sale, and the copper extract is prepared so it can be directed to its refining at División Ventanas. In the copper filtration process, it may occur an important number of non-programmed stops, which is why it is fundamental to work on improving the current maintaining cycle by the prediction of failures to avoid loss in the productivity.

As an improvement in the maintaining cycle in the process of Larox filtration, firstly a review of the reported failures it is made during January-June 2021 to carry out a maintenance study focused on reliability (RCM, Reliability Centered Maintenance) which it is used as a basis to the project. Afterwards, by the SOMA (Sistema Operativo de Mantenimiento – Maintenance Operating System) of Codelco, it is evaluated the criticality of each failure included in the RCM, for the purpose of checking which are the most critical in economic, environmental and security standards.

The Larox filtration process has a report system in SAP, whereby the operators, maintainers and engineers report anomalies detected during the running of the devices and installations. These warnings are downloaded in Excel format, which are catalogued by level of importance; low, medium, or high depending of the data obtained in the study of criticality made previously. Once the available warns are categorized (around 420 warns), an algorithm of Natural Language Processing it is programmed in Python, an area of Artificial Intelligence accountable for understanding the way humans communicate to categorize new entered warns after training the algorithm with the initial database. In this algorithm, firstly a data cleansing it is made, in order to be pre-processed afterwards and try different classification methods, searching for the one that obtains the highest accuracy percentage of precision.

Once the most accurate algorithm is obtained, this one is used to program a new code in Python, which receives a new database downloaded from SAP. It processes the new loaded warns and it delivers a table with the respective level of importance predicted for each one of them. The final algorithm possesses an accuracy of approximately 80 %.

Índice de Contenidos

1. Introducción	1
2. Objetivos	2
2.1. Objetivo General	2
2.2. Objetivos Específicos	2
3. Contexto operacional de la empresa	3
3.1. Antecedentes de la empresa	3
3.1.1. Descripción de la empresa	3
3.1.2. División Andina	3
3.2. Problemática de la empresa	4
3.3. Proceso de División Andina	4
3.3.1. Explotación	4
3.3.1.1. Mina rajo	4
3.3.1.2. Mina subterránea	5
3.3.2. Concentradora	5
3.3.2.1. Chancado Don Luis	5
3.3.2.2. Chancado grueso	5
3.3.2.3. Molienda	5
3.3.2.4. Filtrado	6
3.3.3. Proceso de Filtrado Larox	6
3.3.4. Espesadores de Concentrado	7
3.3.5. Harnero de Limpieza	7
3.3.6. Estanque Homogenizador	7
3.3.7. Filtro Larox	8
3.3.8. Correas Transportadoras	9
3.4. Mantenimiento en Codelco	9

3.4.1. Política corporativa del mantenimiento	9
3.4.2. Sistema Operativo de Mantenimiento (SOMA)	10
3.4.2.1. Áreas del Mantenimiento	10
3.4.3. SAP	13
4. Marco Teórico	14
4.1. Mantenimiento Centrado en Confiabilidad (RCM)	14
4.1.1. Conceptos clave de un RCM	15
4.2. Machine Learning	18
4.2.1. Tipos de aprendizaje	18
4.2.1.1. Aprendizaje supervisado	18
4.2.1.2. Aprendizaje no supervisado	19
4.2.1.3. Aprendizaje por refuerzo	19
4.2.2. Técnicas de Machine Learning para aprendizaje supervisado	20
4.2.2.1. Máquina de Soporte Vectorial	20
4.2.2.2. Árboles de Decisión	21
4.2.2.3. Bosques Aleatorios	22
4.2.2.4. Naïve Bayes	24
4.2.2.5. K vecinos más cercanos	24
4.2.3. Métricas de evaluación	26
4.2.3.1. Precisión (Precision)	26
4.2.3.2. Exhaustividad (Recall)	26
4.2.3.3. Exactitud (Accuracy)	27
4.2.3.4. Valor F1 (F1-score)	27
4.2.4. Procesamiento del Lenguaje Natural	28
4.2.4.1. Tf-idf (Term Frequency - Inverse Document Frequency)	28
4.3. Python	29
4.3.1. Jupyter Notebook	29
4.3.2. Librerías utilizadas	29
5. Metodología	31
5.1. Visualización de avisos	31
5.2. Revisión fallas	32
5.2.1. Revisión frecuencia de fallas	32
5.2.2. Análisis de planes de mantenimiento actuales	35
5.3. Revisión de avisos	37
5.3.1. Categorización de avisos de entrenamiento	37

5.3.2. Programación de algoritmo de pruebas	38
5.3.2.1. Limpieza de base de datos	38
5.3.2.2. Preprocesado de información	39
5.3.2.3. División de datos para entrenamiento y prueba	40
5.3.2.4. Prueba de Máquina Soporte Vectorial	40
5.3.2.5. Prueba de Árboles de Decisión	42
5.3.2.6. Prueba de Bosques Aleatorios	44
5.3.2.7. Prueba de Naïve Bayes	45
5.3.2.8. Prueba de KNN	45
5.3.3. Programación de algoritmo final	46
5.3.3.1. Carga de librerías y bases de datos	46
5.3.3.2. Generación de reporte de datos	48
5.3.3.3. Programación de algoritmo de predicción	49
5.4. Análisis de resultados	53
6. Conclusiones	54
Referencias	56
A. Anexo I: Categorización avisos inicial	58
B. Anexo II: Resultado de predicción	59
C. Anexo III: RCM	60
D. Anexo IV: Código de prueba de técnicas	61
E. Anexo V: Código final	74

Índice de Tablas

3.1. Equipos perforación Mina Rajo	4
3.2. Equipos carguío y transporte Mina Rajo	4
3.3. Equipos carguío y transporte Mina Subterránea	5
4.1. Matriz de confusión	26
5.1. Fallas reportadas en Filtro Larox	36
5.2. Fallas reportadas en bombas	36
5.3. Fallas reportadas en cintas transportadoras	36
5.4. Fallas reportadas en Espesadores de Concentrado	37
5.5. Fallas reportadas en Harnero Delkor	37
5.6. Resultado f1-score para distintas técnicas utilizadas	47

Índice de Figuras

3.1. Proceso de Filtrado Larox.	6
3.2. Diagrama de Harnero Delkor. Fuente: Delkor	7
3.3. Paquete de placas de filtro completo. Fuente: Flowrox	9
3.4. Capa 2: Ciclo de Mantenimiento. Fuente: SOMA	13
4.1. Diagrama de aprendizaje supervisado. Fuente: neurospace.io	18
4.2. Diagrama de aprendizaje no supervisado. Fuente: techvidvan.com	19
4.3. Diagrama de aprendizaje por refuerzo. Fuente: medium.com	19
4.4. Frontera de decisión SVM. Fuente: analisisdedatos.net	20
4.5. Ejemplo de relación entre C y Gamma. Fuente: runebook.dev	21
4.6. Estructura árbol de decisión. (10)	22
4.7. Ejemplo de funcionamiento de Bosques Aleatorios. (11)	23
4.8. Ejemplo de funcionamiento KNN. Fuente: datacamp.com	25
5.1. Acceso a transacción iw29.	31
5.2. Campos para completar.	32
5.3. Visualización de avisos.	32
5.4. Limpieza de base de datos y cantidad de fallas.	33
5.5. Ubicación técnica abreviada y su denominación.	33
5.6. Cantidad de fallas para cada ubicación.	34
5.7. 15 ubicaciones técnicas con más fallas reportadas.	34
5.8. Extracto de base de datos de entrenamiento.	37
5.9. Información del tamaño base de datos de entrenamiento.	38
5.10. Definición de variables X e y.	39
5.11. Código para el preprocesamiento de la información.	39
5.12. Vectorización TF-IDF.	40
5.13. División de datos en entrenamiento y prueba.	40

5.14. Importación de librerías de SVM.	41
5.15. GridSearch para SVM.	41
5.16. Predicción de SVM.	42
5.17. Testing Accuracy para Árboles de Decisión.	43
5.18. Predicción de Árboles de Decisión.	43
5.19. Librerías y GridSearch en Random Forest.	44
5.20. Resultado de Random Forest.	44
5.21. Definición del algoritmo de Naïve Bayes.	45
5.22. Matriz de confusión algoritmo de Naïve Bayes.	45
5.23. Ciclo “for” para evaluar distintos valores de K.	46
5.24. Gráfica de precisión con K vecinos.	46
5.25. Librerías requeridas para el algoritmo.	47
5.26. Carga de bases de datos.	47
5.27. Fragmento de imagen de ubicaciones y avisos entregado por el algoritmo.	48
5.28. Imagen entregada por el algoritmo con ubicaciones técnicas con más avisos.	48
5.29. Definición de variables.	49
5.30. Limpieza de variables X.	49
5.31. Vectorización de variables.	50
5.32. Programación de Bosques Aleatorios.	50
5.33. Programación de DataFrame “data”.	51
5.34. Obtención de DataFrame final.	51
5.35. Obtención de imagen final.	52
5.36. Tabla comparativa.	53

1 | Introducción

La creciente alza durante los últimos años en investigación sobre Inteligencia Artificial y la comunicación hombre-máquina ha generado que las empresas productivas tengan interés en utilizar estas tecnologías para sus procesos, gestión, recursos humanos, mantenimiento, entre otras áreas.

Codelco es una de las mayores empresas extractoras y exportadoras de cobre del mundo, por lo que mejorar constantemente sus procesos productivos es una tarea que realizan día a día.

El objetivo de este trabajo de título es, en primer lugar, analizar las fallas que han ocurrido en el período de un semestre en la planta de filtrado Larox de Codelco división Andina mediante un estudio de mantenimiento centrado en confiabilidad, para posteriormente verificar que fallas reportadas no poseen planes de mantenimiento, para gestionar la creación de uno. Luego de este estudio previo se utilizan los avisos descargados de SAP de las fallas reportadas para entrenar un algoritmo de Procesamiento del Lenguaje Natural, el cual es capaz de procesar nuevos avisos reportados y asignarles un nivel de importancia a cada uno de “alto”, “medio” y “bajo” según corresponda.

Se debe realizar un análisis en profundidad de cada falla y su potencial riesgo, por lo que se utiliza el Sistema Operativo del Mantenimiento (SOMA) que es la base del ciclo de mantenimiento de Codelco, para evaluar la criticidad de cada falla y el impacto que tiene para la seguridad de las personas, el medio ambiente, la producción y los equipos.

2 | Objetivos

2.1. Objetivo General

Desarrollar una mejora al actual plan de mantenimiento de la línea de filtrado Larox de Codelco división Andina, la cual se hace mediante la técnica de inteligencia artificial llamada Procesamiento del Lenguaje Natural (NLP). Se plantea comenzar desarrollando un Mantenimiento Centrado en Confiabilidad (RCM) que se utilizará como base para proponer un plan de mantenimiento anual, el cual se ajusta con el algoritmo a medida que se reporten fallas no planificadas.

2.2. Objetivos Específicos

- Revisar las fallas no planificadas reportadas, estableciendo su frecuencia de ocurrencia.
- Hacer una revisión de los planes actuales disponibles para los equipos de la planta de filtrado Larox y verificar que fallas no planificadas carecen de plan de mantenimiento, para que el equipo de confiabilidad los defina.
- Desarrollar un RCM para los distintos equipos de la planta de filtrado Larox en base a las fallas reportadas.
- Aplicar técnicas de NLP para facilitar la elaboración de reportes y sugerir la asignación de tareas de mantenimiento a realizar.

3 | Contexto operacional de la empresa

3.1. Antecedentes de la empresa

3.1.1. Descripción de la empresa

Codelco es una empresa perteneciente al Estado de Chile y esta catalogada como una de las mayores productoras de cobre de mina del mundo. Su negocio implica la exploración, desarrollo y explotación de recursos mineros, los cuales posteriormente son procesados para producir concentrado de cobre, cátodos de cobre y subproductos. La operación de Codelco se realiza a través de ocho divisiones: Chuquicamata, Ministro Hales, Radomiro Tomic, Gabriela Mistral, Salvador, Andina, Teniente y Ventanas. A su vez, toda la estrategia corporativa es dirigida desde Casa Matriz, ubicada en Santiago de Chile. (1)

3.1.2. División Andina

La división Andina opera el yacimiento Río Blanco, cuya riqueza es conocida desde 1920, pero recién en 1970 se inicia su explotación. Andina está ubicada a 80 kilómetros al noreste de Santiago, entre 3.700 y 4.200 metros sobre el nivel del mar (msnm). Actualmente se realiza la explotación en la mina subterránea de Río Blanco y en la mina a rajo abierto Sur Sur.

Andina produce aproximadamente 210.000 toneladas métricas anuales de concentrado de cobre 2.000 toneladas métricas de molibdeno. El concentrado de cobre es dirigido a división Ventanas para su posterior fundición y refinación, logrando así la obtención de cátodos de cobre con un 99,9 % de pureza. (2)

3.2. Problemática de la empresa

Actualmente varias empresas manejan grandes volúmenes de datos, los cuales pueden contener información valiosa que no siempre se utiliza. En el caso de Codelco Andina, toda la gestión del mantenimiento se realiza vía SAP PM. Durante la operación de Codelco ocurren fallas en los equipos, las cuales son reportadas mediante un aviso en SAP, el cual posteriormente personal de planificación del mantenimiento le asigna una prioridad para que la falla sea solucionada. En los turnos de trabajo se genera una gran cantidad de avisos, por lo que los planificadores comentan que durante su horario laboral no siempre alcanzan a revisarlos uno por uno y asignarles el nivel de importancia. Debido a esta problemática, es que el equipo de mantenimiento de la Planta de Filtrado solicita un algoritmo que sea capaz de procesar estos avisos y asignarles el nivel de prioridad que requieren automáticamente.

3.3. Proceso de División Andina

3.3.1. Explotación

En división Andina, para el proceso de explotación se cuenta con dos fuentes de extracción, la Mina a Rajo y la Mina Subterránea.

3.3.1.1. Mina rajo

La mina rajo en primer lugar realiza la perforación, posteriormente se efectúa proceso de tronadura y carguío, para pasar a la planta concentradora. Los equipos que se tienen son:

Tabla 3.1: Equipos perforación Mina Rajo

Equipos	Marca	Cantidad
Perforadora primaria eléctrica	Epiroc	2
Perforadora primaria diésel	Caterpillar	1
Perforadora primaria diésel	Epiroc	1
Perforadora secundaria diésel	Epiroc	2

Tabla 3.2: Equipos carguío y transporte Mina Rajo

Equipos	Marca	Cantidad
Pala cable eléctrica	P&H	2
Pala hidráulica diésel	Komatsu	2
Cargador frontal diésel	Letorneau	2
CAEX diésel	Epiroc	33

3.3.1.2. Mina subterránea

El proceso de mina subterránea consiste en un proceso de hundimiento, producción, transporte con equipos 5 Load Haul Dump (LHD), piques de traspaso, buzones y posteriormente el transporte en camiones CAEX hacia la planta concentradora. Los equipos que se utilizan son:

Tabla 3.3: Equipos carguío y transporte Mina Subterránea

Equipos	Marca	Cantidad
LHD	Caterpillar	9
CAEX MS	Atlas Copco	9
Tolva sur	-	1
Alimentador	-	2
Correa 72B	-	1

3.3.2. Concentradora

El proceso de la concentradora se realiza a través de dos flujos, el que comienza por el Chancado Don Luis y recibe el mineral proveniente de la Mina Rajo, y Chancado grueso que trata el mineral de la Mina Subterránea. Luego, el cobre de ambos procesos es dirigido al proceso de flotación, para luego pasar al filtrado.

3.3.2.1. Chancado Don Luis

El mineral proveniente de la Mina Rajo, es dirigido al Chancador Don Luis, el cual reduce el tamaño de las rocas procesadas, para pasarlas al prechancado y molinos SAG.

3.3.2.2. Chancado grueso

El mineral proveniente de la mina subterránea, es dirigido al Chancado grueso y chancado fino para reducir su tamaño, luego es dirigido al proceso de molienda convencional.

3.3.2.3. Molienda

En el proceso de molienda el mineral pasa por dos molinos de bolas, para luego dirigirse a cuatro bombas ciclón y cuatro baterías hidrociclón. Finalmente, la materia prima es dirigida a las celdas de flotación de la planta de filtrado.

3.3.2.4. Filtrado

Para comenzar el proceso de filtrado, en primer lugar el mineral pasa por el proceso de flotación a través de las Celdas Rougher, para luego dirigirse a los dos procesos de filtrado que se realizan en paralelo, el filtrado convencional que se realiza a través de filtros cerámicos y un tubo de secado, y el filtrado Larox, que al ser relevante para el tema en estudio se explica con mayor detalle en el punto 3.3.3.

3.3.3. Proceso de Filtrado Larox

En la División Andina de Codelco se obtienen principalmente dos productos: El concentrado de cobre y el molibdeno. Para la obtención de estos productos es necesario pasar por procesos vistos de explotación, chancado, molienda y flotación convencional, obteniéndose un concentrado mixto de Cobre y Molibdeno, el cual es dirigido a la Planta de Molibdeno y Filtrado de Cobre para ser procesado y posteriormente despachado. Mediante flotación selectiva son separados el molibdeno del cobre, siendo este último dirigido a dos procesos de filtrado que trabajan en paralelo, el Filtrado Convencional y el Filtrado Larox (Figura 3.1). El Filtrado Larox tiene como objetivo filtrar el cobre y reducir su humedad hasta un rango de 9,0 % - 9,2 %, para posteriormente ser envasado y despachado a la División Ventanas.

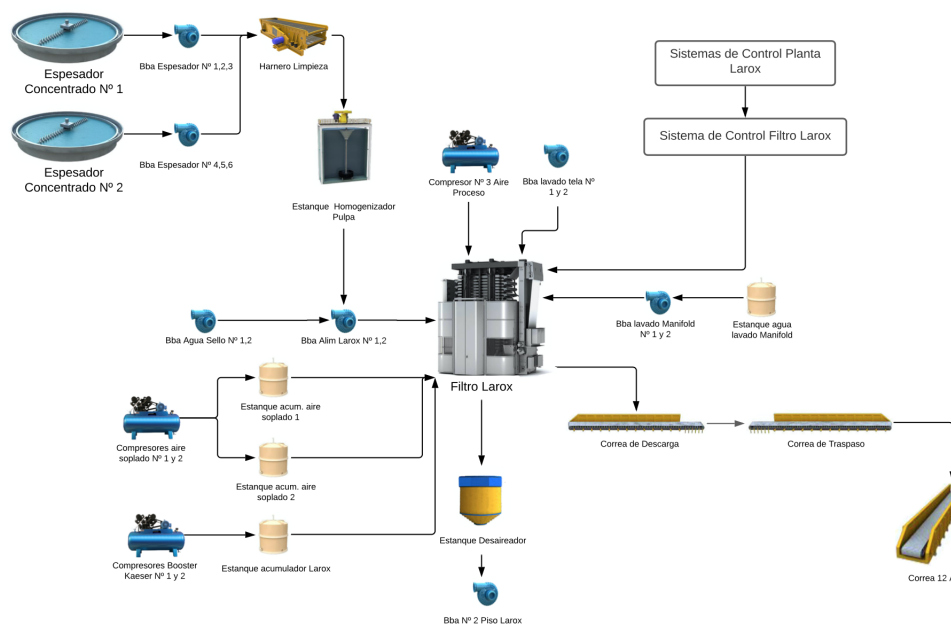


Figura 3.1: Proceso de Filtrado Larox.

Para este proceso, se utilizan los siguientes equipos:

3.3.4. Espesadores de Concentrado

Actualmente en la Planta de Molbideno y Filtrado de Cobre (PMFC) se cuenta con 2 espesadores de concentrado, desde los cuales proviene, como su nombre lo indica, concentrado de cobre con porcentaje de sólidos que varía entre un 50 % - 70 %, cual es dirigido a través de bombas al Harnero de Limpieza.

3.3.5. Harnero de Limpieza

El concentrado de cobre proveniente de los espesadores de concentrado se recibe por el harnero de limpieza, el cual es de modelo Delkor 13C/10-15S (Figura 3.2). Su principal función es eliminar los sólidos que pudieran dañar al filtro en los procesos siguientes. El harnero dirige el concentrado de cobre al Estanque Homogenizador.

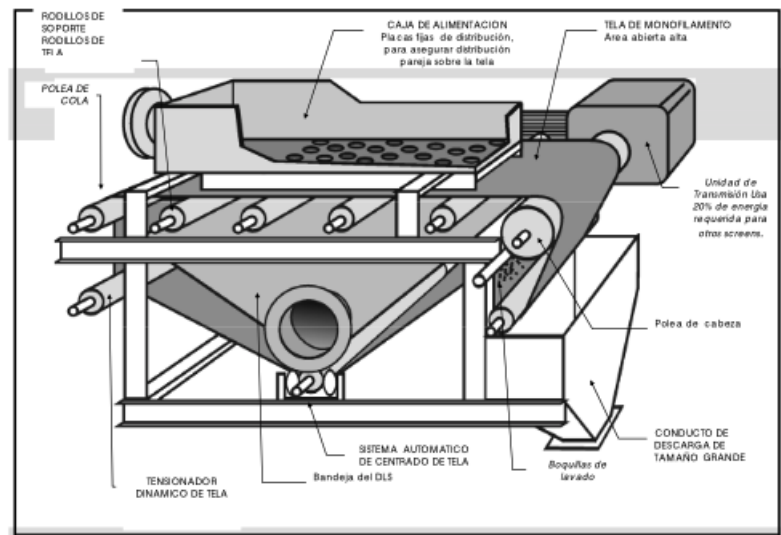


Figura 3.2: Diagrama de Harnero Delkor. Fuente: Delkor

3.3.6. Estanque Homogenizador

El estanque homogenizador en su interior posee un agitador, quien tiene como objetivo mantener el concentrado de cobre en constante movimiento, con el fin de que éste último se aglomere y detenga la fluidez del proceso. Posteriormente mediante bombas se alimenta al filtro Larox con el concentrado de cobre.

3.3.7. Filtro Larox

Es un filtro de placas (también denominado de alta eficiencia), el cual contiene 24 placas verticales (Figura 3.3), y su objetivo es que el concentrado de cobre alcance la humedad y porcentaje de limpieza requeridos. El proceso de filtrado consta de 4 partes:

1. Filtración: Esta técnica de separación consiste en hacer pasar la “pulpa” (mezcla de sólidos y líquidos) a través de una tela filtrante, para así retener la mayoría de los componentes sólidos de la mezcla separándolos de los líquidos.
2. Prensado: Se realiza mediante el uso de aire a alta presión (12 [bar] - 13 [bar]), el cual ingresa mediante uno de los costados, a través de mangueras a todas las cámaras del paquete de placas del filtro. El aire comprime la “torta” (concentrado de cobre con consistencia viscosa) contra la superficie de la tela ayudando a la evacuación del agua a través de rejillas, dirigida a colectores y finalmente evacuada del filtro.
3. Secado: Para el proceso de secado se utiliza aire a presión (5 [bar] - 6 [bar]) donde con ayuda de una válvula se regula la entrada del aire de secado. El aire pasa a través de los tubos de alimentación de la pulpa, llenando las cámaras del filtro y haciendo subir el diafragma. El flujo de aire logra por un lado desprender la “torta” de los diafragmas llevándolos a su posición inicial, y por otro lado alcanzar el porcentaje de humedad requerido de la “torta”.
4. Descarga: Consiste en el desbloqueo del paquete de placas del filtro, retirando los pasadores de las 4 columnas. Mediante los cilindros de acción rápida (QAC) se realiza la apertura del paquete de placas, luego la tela comienza a moverse para permitir la salida de la “torta” por ambos lados del filtro. Simultáneamente se realiza el lavado de la tela mediante agua a presión por ambos lados.

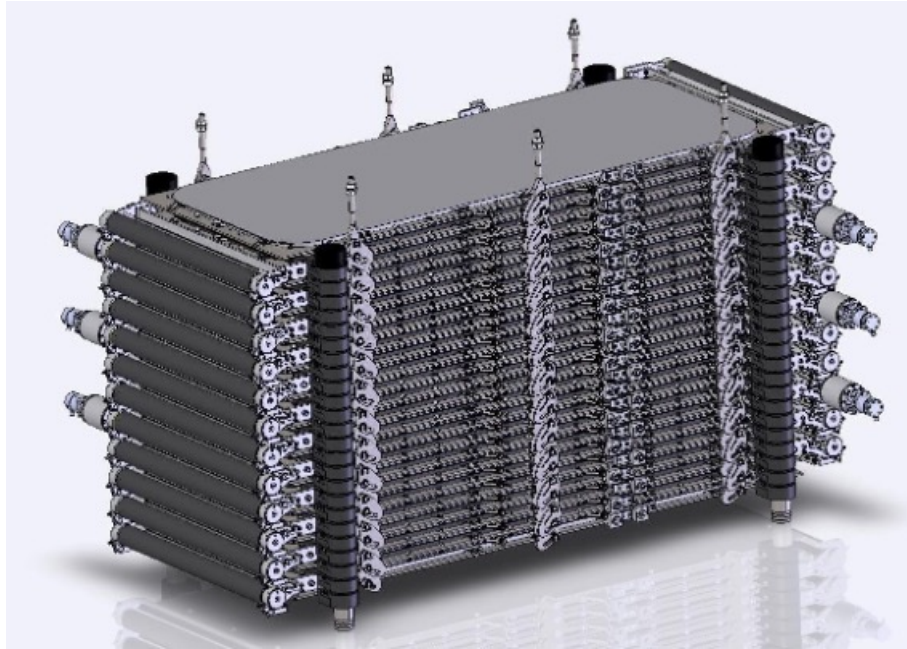


Figura 3.3: Paquete de placas de filtro completo. Fuente: Flowrox

3.3.8. Correas Transportadoras

Al final del proceso de filtrado Larox se encuentran las correas transportadoras, que están encargadas de transportar el concentrado de cobre al tren que se dirige a Codelco División Ventanas para continuar con el proceso productivo. En el proceso se utilizan 3 correas transportadoras, las cuales son:

- Correa de Descarga: Cinta de 72", la cual esta compuesta de 4 telas y tiene un largo de 44 metros.
- Correa de Traspaso: Cinta de 36", compuesta de 2 telas y con largo de 34 metros.
- Correa 12-A: Cinta de 36", compuesta de 3 telas y con largo de 127 metros.

3.4. Mantenimiento en Codelco

3.4.1. Política corporativa del mantenimiento

En Codelco el mantenimiento se rige bajo una política a nivel corporativo, la cual busca asegurar la confiabilidad de los activos, maximizar los coeficientes operacionales y dar sustentabilidad a los planes de producción, para esto se implementan estándares y planes de mejora en los ámbitos de procesos, sistemas, personas y cultura. La política corporativa del mantenimiento se rige bajo 9 puntos, los cuales son (3):

1. Cumplir las políticas corporativas vigentes, procedimientos, normas, estándares y reglamentación dentro de un marco ético, integro, de confianza y de respeto al resto de la organización y sus terceros.
2. Actualizar estrategias y planes de mantenimiento continuamente con foco en mejoras de productividad.
3. Promover la mejora continua a través de la implementación del sistema operativo C+, definiendo y actualizando estándares operacionales, detectando oportunamente nuestras brechas, promoviendo la estandarización y las mejores practicas de la industria.
4. Programar planes de mantenimiento coherentes con los planes de producción, y ejecutarlos con adherencia, asegurando efectividad en el cumplimiento de planes y eficiencia en el uso de los recursos.
5. Tener la estructura organizacional y competencias para asegurar los roles, funciones y responsabilidades de todas y todos los trabajadores de las áreas de mantenimiento de acuerdo a los roles definidos corporativamente.
6. Medir y gestionar continuamente el grado de madurez del mantenimiento con el objetivo de asegurar el desarrollo hacia un mantenimiento proactivo.
7. Generar instancias de participación activa que permitan asegurar acciones estandarizadas de mejora del mantenimiento y transferencias de mejores practicas entre las áreas y divisiones.
8. Utilizar el sistema SAP como herramienta oficial y única para la administración de la gestión del mantenimiento.
9. Impulsar el uso de tecnologías que aporten al mejoramiento de la gestión del mantenimiento en todos sus ámbitos.

3.4.2. Sistema Operativo de Mantenimiento (SOMA)

El Sistema Operativo de Mantenimiento (SOMA) se define como un conjunto de estándares y procesos que apuntan a optimizar la gestión de activos, donde su objetivo es generar valor a Codelco a través de los activos, asegurando el cumplimiento de estrategias de mantenimiento que optimicen su rendimiento.

(4)

3.4.2.1. Áreas del Mantenimiento

Dentro del departamento de mantenimiento en Codelco, existen tres sub-divisiones establecidas por el SOMA (4), las cuales velan por la correcta gestión de los activos y deben estar sincronizadas entre sí. Las áreas existentes son:

1. **Confiabilidad:** El objetivo de esta área es optimizar el ciclo de vida del activo, asegurando así la capacidad de este de cumplir adecuadamente con las funciones para las que fue diseñado bajo el contexto operacional en el que se encuentra inmerso. Dentro de las funciones del área de confiabilidad, se considera la de establecer estrategias de mantenimiento óptimas para los equipos, así como mejorar su disponibilidad y confiabilidad. Además, están encargada de la incorporación de buenas prácticas operacionales y monitoreo de los activos, para disminuir y evitar las actividades no programadas.
2. **Planificación:** El área de planificación cumple el objetivo general de generar y habilitar los planes de mantenimiento basados en las estrategias construidas por confiabilidad, con el fin de maximizar la continuidad operacional y utilizar de forma eficiente el tiempo, recursos y materiales de trabajo. Dentro de las funciones de planificación se encuentra la de preparar los planes de mantenimiento y verificar su óptima ejecución, gestionar el flujo de trabajo de las actividades, administrar los recursos para la ejecución (repuestos y contratos), y también la coordinación con las áreas operativas.
3. **Ejecución:** El objetivo del área ejecutora es el de llevar a cabo de forma eficaz y eficiente las actividades del programa de mantenimiento, respetando los tiempos y recursos, gestionando en conjunto a confiabilidad las actividades no planificadas que vayan surgiendo durante la operación. Dentro de sus funciones se considera la de ejecutar las ordenes de trabajo según los procedimientos establecidos, cumpliendo la calidad y seguridad requerida, participar en el ciclo de planificación, reportar los hallazgos visualizados durante la ejecución de las actividades, y notificar el cierre de las órdenes de trabajo finalizadas.

El SOMA esta dividido en 5 capas enumeradas desde la 0 a la 4, que definen los estándares de gestión de mantenimiento de Codelco, las cuales son:

- **Capa 0. “Política del mantenimiento”:** La primera capa del SOMA establece el propósito de la empresa, el cual es asegurar la confiabilidad de los activos, maximizar los indicadores productivos y dar sustentabilidad a sus operaciones. Dentro de esta política, se tienen 9 reglas (4):
 1. Cumplir las políticas corporativas vigentes, procedimientos, normas, estándares y reglamentación.
 2. Actualizar estrategias y planes de mantenimiento continuamente con foco en mejoras de productividad.
 3. Promover la mejora continua a través de la implementación del sistema operativo C+.
 4. Programar planes de mantenimiento coherentes con los planes de producción, y ejecutarlos con adherencia.
 5. Tener la estructura organizacional y competencias para asegurar los roles, funciones y responsabilidades.

6. Medir y gestionar continuamente el grado de evolución del mantenimiento.
 7. Generar instancias de participación activa entre las diferentes áreas operativas.
 8. Utilizar el sistema SAP.
 9. Impulsar el uso de tecnologías que aporten al mejoramiento de la gestión del mantenimiento.
- Capa 1. “Fases del Sistema Operativo”: El SOMA se compone de 8 fases, las cuales promueven el cumplimiento de los objetivos de la corporación (4). Estas 8 fases son:
 1. Fase de inversión en proyectos.
 2. Estudio de criticidad y metas de desempeño.
 3. Plan estratégico y plan matriz.
 4. Ciclo del mantenimiento.
 5. Decisiones de fin de vida útil.
 6. Sinergias con áreas relacionadas.
 7. Sistemas de gestión, estructura organizacional y competencias.
 8. Análisis de desempeño.
 - Capa 2. “Ciclo del Mantenimiento”: El ciclo de mantenimiento esta conformado por 5 etapas (Figura 3.4). El presente trabajo, presenta un aporte a las tareas de Identificación y priorización (Jerarquizando la importancia de cada falla reportada) y de Post-mantenimiento (Aportando con la elaboración de reportes).

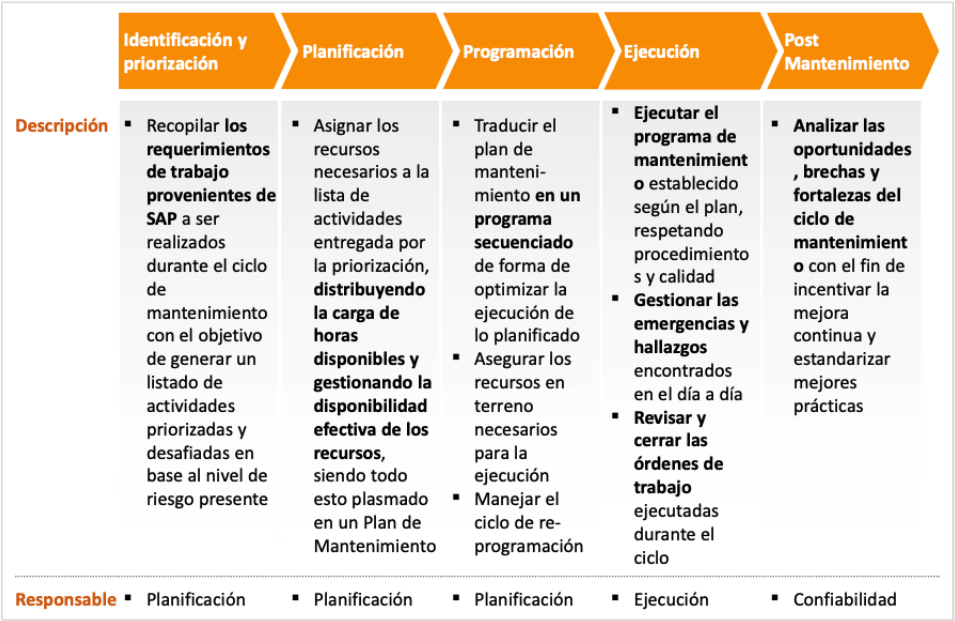


Figura 3.4: Capa 2: Ciclo de Mantenimiento. Fuente: SOMA

- Capa 3. “Proceso”: El SOMA establece que cada proceso se compone de un flujograma, el cual detalla las etapas a realizar.
- Capa 4. “Estándar”: Esta capa indica que las etapas relevantes para la corporación estén estandarizadas, lo cual garantiza que siempre se haga de igual manera.

3.4.3. SAP

SAP es el software utilizado en Codelco para la gestión de procesos, mantenimiento y logística. Este programa aporta con la centralización de los datos de Codelco a nivel corporativo, evitando la duplicidad de datos y reduciendo errores y costos de almacenamiento de la información. En el área mantenimiento se utiliza principalmente el módulo PM (Plant Maintenance), el cual permite la creación de avisos, ordenes de trabajo y notificación post-mantenimiento.

4 | Marco Teórico

4.1. Mantenimiento Centrado en Confiabilidad (RCM)

El Mantenimiento Centrado en Confiabilidad (6) o RCM (Reliability Centered Maintenance) es una metodología de mantenimiento que busca mejoras significativas en el rendimiento de la planta y confiabilidad del equipo al que se le es aplicada, mediante la búsqueda de la continuidad operacional del equipo en su contexto operativo. El origen de esta técnica se remonta a los años 60, la cual ha evolucionado con el pasar de los años. Sus primeras aplicaciones en la industria aeronáutica bajo el nombre de método MSG3 fueron cruciales para el perfeccionamiento de esta metodología.

La sociedad americana de ingenieros automotrices, al encontrar potencial en la metodología del RCM publicó la norma SAE JA1011 en el año 1999, la cual define un estándar para los criterios de evaluación a procesos de mantenimiento (5).

El RCM se describe mediante la respuesta a las denominadas "Siete preguntas básicas", las cuales son:

1. ¿Cuáles son las funciones y los estándares de desempeño asociados del activo en su contexto operativo actual?
2. ¿En que aspecto no responde al cumplimiento de sus funciones?
3. ¿Qué ocasiona cada falla funcional?
4. ¿Qué sucede cuando se produce cada falla en particular?
5. ¿De qué modo afecta cada falla?
6. ¿Qué puede hacerse para predecir o prevenir cada falla?
7. ¿Qué debe hacerse si no se encuentra el plan de acción apropiado?

4.1.1. Conceptos clave de un RCM

Para responder a las preguntas mencionadas, es que se definen algunos conceptos claves, los cuales son:

- **Funciones y niveles de desempeño:** El primer paso del RCM es definir las funciones de cada equipo dentro del contexto operativo, como así también los estándares de desempeño requeridos. Dentro de las funciones, se encuentran dos categorías:
 1. **Funciones primarias:** Que sintetizan el motivo por el que fue adquirido el bien en una primera instancia, dentro de estas se encuentran temas relacionados a rendimiento, capacidad de producción o transporte, calidad del producto, velocidad, entre otros.
 2. **Funciones secundarias:** Aquellas funciones que se esperan que los equipos cumplan más allá de su función primaria, como seguridad, control, eficiencia de operación, cumplimiento con las normas medioambientales, o inclusive la estética del bien.
- **Fallas funcionales:** Se define como falla funcional cuando un bien es incapaz de cumplir una función a un nivel de desempeño aceptable por el usuario (6), por ende, para poder aplicar herramientas apropiadas en el manejo de fallas es necesario identificar cuales pueden presentarse, es por esto que el RCM lo hace en dos niveles:
 - Identificando que circunstancias provocaron la falla.
 - Identificando que situaciones causan esas circunstancias.
- **Modos de falla:** Una vez identificada las fallas funcionales, el siguiente paso es identificar los hechos que de manera razonablemente posible puedan haber causado la falla, los cuales se denominan modos de falla. En estos modos de falla están incluidos aquellos que han ocurrido en equipos que operan en el mismo contexto del que está en estudio, las fallas que se previenen con los planes de mantenimiento actuales y las fallas que no han ocurrido pero que es esperable que puedan suceder. Se deben considerar tanto fallas ocurridas por el error humano como aquellas ocurridas por error de diseño.
- **Efectos de Falla:** Luego de analizar los modos de falla, el siguiente paso es hacer un listado con los efectos de falla, en los cuales se menciona que ocurre con cada modo de falla. Dentro de la descripción de lo que ocurre, se deben considerar aspectos tales como:
 - Evidencias de que la falla ha ocurrido.
 - De que modo se representa amenaza para la seguridad.
 - De que modo se representa amenaza para el medio ambiente.

- Como afecta a la producción y operaciones.
 - Daños materiales causados por la falla.
 - Como se debe reparar la falla.
- Consecuencias de la falla: En el RCM es de gran importancia realizar un análisis de la consecuencia que trae cada falla, ya que cada una de estas afecta a la organización en un nivel diferente. Si esta consecuencia es alta a nivel productivo, monetario, de seguridad o de medio ambiente, se le da una mayor importancia o recursos para prevenir que sucedan. Sin embargo, si la falla tiene nulas bajas consecuencias, es probable que no se le de mayor importancia a evitar su ocurrencia. En el RCM se clasifican estas consecuencias en cuatro principales grupos, los cuales son:
1. Consecuencias de fallas ocultas: Son aquellas que no tienen un impacto directo, pero pueden causar que la organización sea expuesta a varias fallas con altas consecuencias.
 2. Consecuencias ambientales y para la seguridad: En la actualidad tanto la seguridad de las personas como del medio ambiente es uno de los focos principales de las empresas, por lo que se consideran aquellas fallas que tienen consecuencias para la integridad de las personas o para normativas ambientales.
 3. Consecuencias operacionales: Si la falla afecta a la producción tanto en calidad, cantidad, atención al cliente o costos operacionales se considera con consecuencias operacionales.
 4. Consecuencias no operacionales: Son aquellas que solo implican un costo directo para su reparación.
- Tareas proactivas: Es el mantenimiento tanto preventivo como predictivo. Son aquellas tareas que se realizan antes de que ocurra la falla para evitar que esto suceda. Estas tareas a su vez se dividen en 3 categorías:
1. Tareas de reacondicionamiento cíclicas: Son aquellas que implica la refabricación o reparación de un componente a un límite de edad establecido, sin darle importancia al estado en ese momento.
 2. Tareas de sustitución cíclicas: Similar a las tareas de reacondicionamiento cíclicas, implican el sustituir un componente en un límite de edad establecido, independiente del estado de éste.
 3. Tareas a condición: Se realiza tanto mantenimiento predictivo como a condición, implicando que los componentes se dejan en servicio a condición de que alcancen los parámetros de funcionamiento deseados.
- Acciones a falta de: Son acciones que se toman cuando no se encuentra una tarea proactiva efectiva y

tratan de forma directa con el estado de falla. Se consideran tres categorías de acciones a falta de:

1. Búsqueda de fallas: Son aquellas tareas que implican la revisión periódica de funciones ocultas para verificar si no se están realizando correctamente.
2. Rediseño: Consiste en la modificación del equipo y procedimientos.
3. Ningún mantenimiento programado: Se deja operar el equipo “a la falla”, por lo que no se tiene un plan de mantenimiento preventivo.

4.2. Machine Learning

El Machine Learning o aprendizaje automático se puede definir como el “subcampo de la inteligencia artificial que proporciona a los computadores la capacidad de aprender sin ser explícitamente programados, es decir, sin que estos necesiten que el usuario indique las reglas que deben seguir para lograr su tarea, sino que la hace automáticamente” (7). Dada la madurez y la versatilidad del Machine Learning (hoy en día puede verse aplicación de algoritmos en áreas como las finanzas, publicidad, robótica, mantenimiento predictivo, entre otros) es que existen diversos enfoques. El Machine Learning a su vez puede agruparse en tres técnicas, las cuales son el aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

4.2.1. Tipos de aprendizaje

4.2.1.1. Aprendizaje supervisado

El aprendizaje supervisado es aquel en donde los datos que utilizamos para la fase de entrenamiento incluyen la solución deseada, la cual es denominada label (etiqueta)(7). En este caso, el objetivo es que en el algoritmo se genere un modelo, el cual una vez le sean entregados datos nuevos (que no haya procesado anteriormente) tenga la capacidad de clasificar correctamente la etiqueta que le corresponde a cada uno.

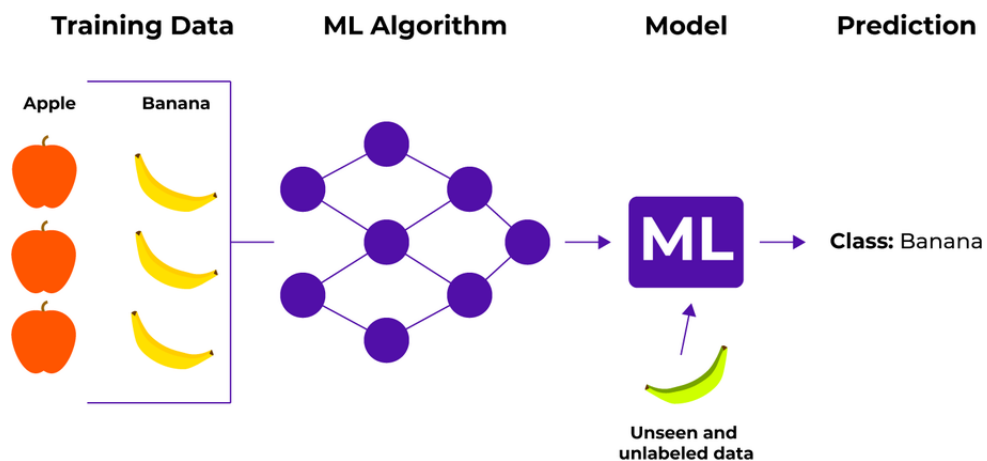


Figura 4.1: Diagrama de aprendizaje supervisado. Fuente: neurospace.io

4.2.1.2. Aprendizaje no supervisado

El aprendizaje no supervisado a diferencia del supervisado, es un algoritmo utilizado cuando la base de datos no posee etiquetas para entrenamiento, y la función principal del modelo es agrupar los datos acorde a patrones que el mismo algoritmo, mediante redes neuronales, sea capaz de decidir.

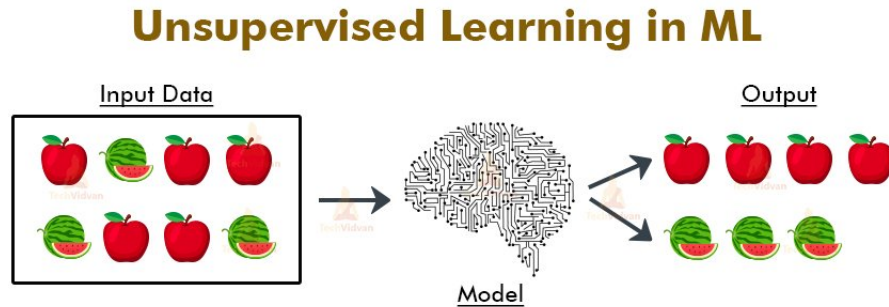


Figura 4.2: Diagrama de aprendizaje no supervisado. Fuente: techvidvan.com

4.2.1.3. Aprendizaje por refuerzo

El aprendizaje por refuerzo es aquel en cual el modelo programado es implementado en forma de un “agente” que interactúa con un entorno definido, y en base a la entrega de recompensas y castigos, el algoritmo es capaz de aprender sobre que decisiones al ser vueltas a tomar le otorgan un beneficio, y cuales no. En la actualidad es ampliamente utilizado en el rubro de simulación, robótica y conducción autónoma.

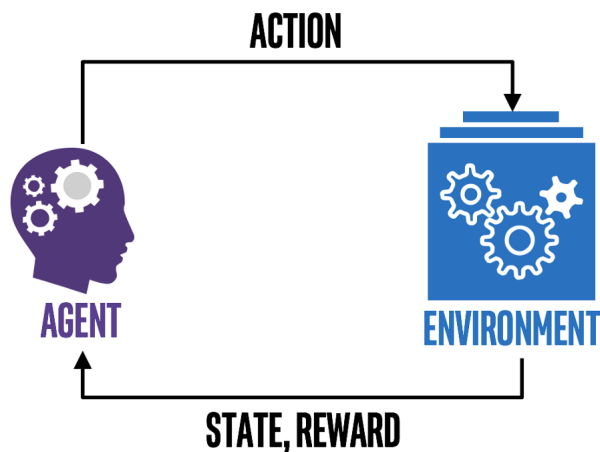


Figura 4.3: Diagrama de aprendizaje por refuerzo. Fuente: medium.com

4.2.2. Técnicas de Machine Learning para aprendizaje supervisado

4.2.2.1. Máquina de Soporte Vectorial

La Máquina de Soporte Vectorial (SVM) es una técnica de Machine Learning de clasificación. Una SVM en primer lugar mapea los datos de entrada a un espacio de características de dimensión mayor (8), para hallar un hiperplano “H” que los separe y maximice el margen “m” entre las clases, como se puede ver en la Figura 4.4:

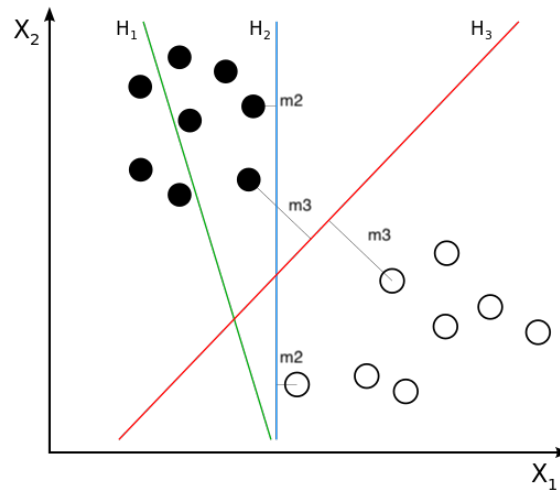


Figura 4.4: Frontera de decisión SVM. Fuente: analisisdedatos.net

Como puede verse en la figura, las tres opciones para el hiperplano “H” representan diferentes características, “H1” no separa las clases en su totalidad, “H2” las separa con un margen “m2” pequeño, mientras que por su parte “H3” representa una mayor distancia “m3” entre los datos y el hiperplano. La definición del hiperplano es usualmente un problema de programación cuadrática, el cual puede ser resuelto introduciendo multiplicadores de Lagrange. La SVM encuentra el hiperplano óptimo utilizando el producto punto con funciones en el espacio de características que son llamadas kernels (8).

Al momento de programar el algoritmo de SVM se definen dos parámetros, los cuales son:

- “C”: Es el parámetro encargado de regular la penalización de los valores que se encuentran fuera de la frontera de decisión. A valores más pequeños, la penalización es baja por lo que el modelo es más conservador y con márgenes más suaves, mientras que para valores más altos, el modelo es mas exigente, con márgenes mas pequeños y se corre el riesgo de caer en un sobre ajuste del modelo (9).
- “gamma”: El parámetro “gamma” genera una relación de los valores mediante una función kernel, lo que puede aumentar la precisión del modelo si se hace una buena relación con el valor “C” 4.5. Para un

“gamma” más bajo se tienen resultados más conservadores, mientras que valores más altos entregan valores más precisos, pudiendo de igual manera caer en un sobre ajuste.

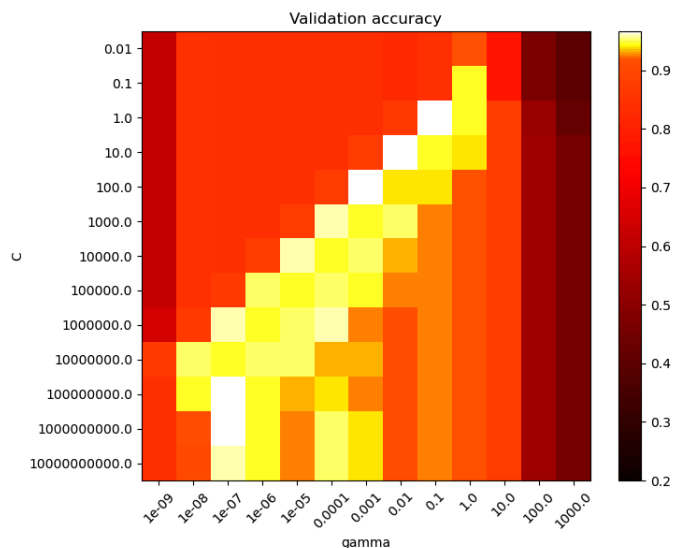


Figura 4.5: Ejemplo de relación entre C y Gamma. Fuente: runebook.dev

4.2.2.2. Árboles de Decisión

Un árbol de decisión es un modelo de predicción que tiene como objetivo el aprendizaje inductivo a partir de observaciones y construcciones lógicas (10). Consiste en categorizar los datos de entrada en base a una serie de condiciones que son definidas durante la fase de entrenamiento del algoritmo, representándose como un árbol conformado por nodos, hojas y ramas (Figura 4.6).

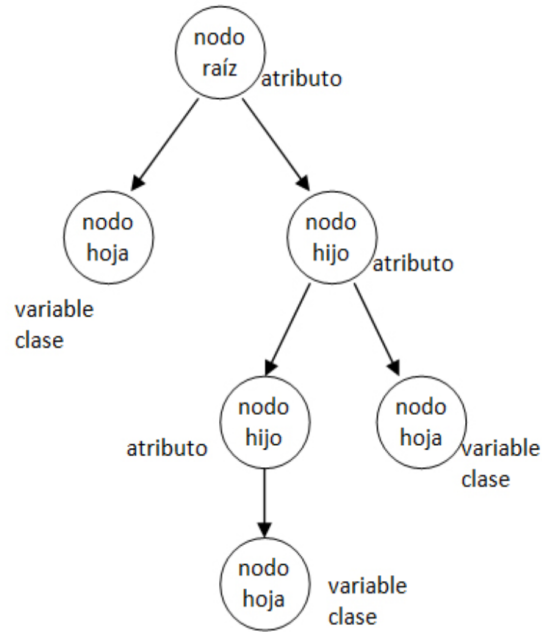


Figura 4.6: Estructura árbol de decisión. (10)

El nodo raíz es el atributo con el cual comienza el proceso de clasificación, los nodos hijo son las posibles respuestas a cada cuestionamiento, mientras que los nodos hoja corresponden a la decisión del algoritmo, la cual coincide con una de las variables clase del problema a resolver (10). Al momento de programar el código de Machine Learning mediante Árboles de Decisión se debe definir el “max_depth”, que corresponde al número máximo de árboles que utilizará el algoritmo para trabajar.

4.2.2.3. Bosques Aleatorios

Los bosques aleatorios son un conjunto de árboles de decisión combinados. Su funcionamiento consiste en que cada árbol de decisión se entrena con una porción de los datos, de esta manera cada árbol aprende de forma distinta al otro, para que así los errores se compensen entre sí y se obtenga una predicción más precisa (11).

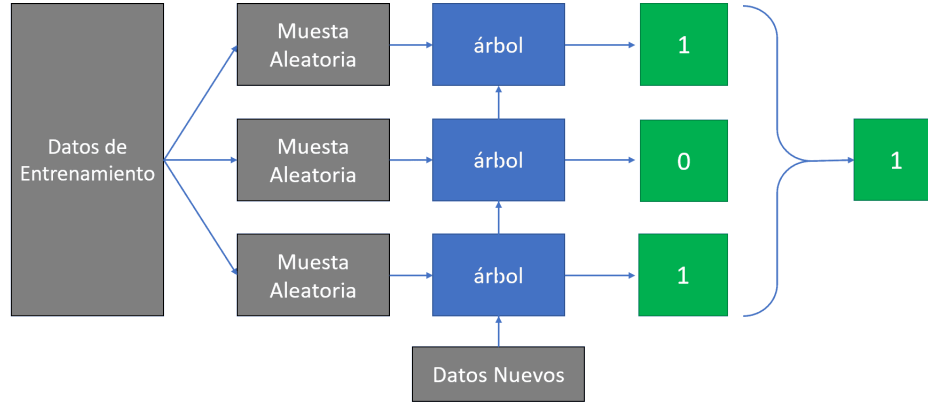


Figura 4.7: Ejemplo de funcionamiento de Bosques Aleatorios. (11)

Como puede verse en el ejemplo de la Figura 4.7 los datos de entrenamiento se dividen en 3 árboles de decisión, en donde cada árbol clasifica un valor y el algoritmo decide mantener el que obtenga más veces una misma predicción.

Los parámetros a regular durante la programación de un algoritmo de bosques aleatorios son:

- “n_estimators”: Número de árboles incluidos en el modelo.
- “max_depth”: Profundidad máxima que pueden alcanzar los árboles.
- “criterion”: Es la medida de homogeneidad que se utiliza para criterio de selección. Usualmente se utilizan en Procesamiento del Lenguaje Natural:
 1. “gini”: Es una medida de las “K” varianzas en el conjunto de clases de un nodo “m”. Se considera una medida de pureza del nodo (12) y se calcula de la forma:

$$G_m = \sum_K^{k=1} p_{mk}(1 - p_{mk}) \quad (4.1)$$

2. “entropy”: Es una forma de cuantificar el desorden de un sistema, a mayor nodos, mayor impureza. Cuando un nodo es puro se puede decir que contiene únicamente observaciones de una clase y posee entropía igual a 0, y si la frecuencia de cada clase es la misma, el valor de entropía alcanza el valor 1 (12). Se calcula de la forma:

$$D = \sum_K^{k=1} p_{mk} * \log(p_{mk}) \quad (4.2)$$

4.2.2.4. Naïve Bayes

El modelo Naïve Bayes consiste en un método estadístico el cual asume que las variables predictoras son independientes entre sí, por lo que se le llama ‘ingenua’. El hecho de tener variables independientes quiere decir que al momento de evaluar un parámetro, los restantes, pese a tener una relación lógica con la variable en estudio, no se tomarán en cuenta. Hacer esta suposición simplifica la capacidad de computo (13). Finalmente, la probabilidad de una hipótesis viene dada por la siguiente fórmula:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \quad (4.3)$$

En donde:

- $P(h|D)$: Probabilidad de la hipótesis h dados los datos D . Es conocida como la probabilidad posterior.
- $P(D|h)$: Probabilidad de los datos D dado que la hipótesis h era cierta. Es conocida como la probabilidad anterior.
- $P(h)$: Probabilidad de que la hipótesis h sea cierta (independiente de los datos), esto es conocido como la probabilidad previa de h .
- $P(D)$: Probabilidad de los datos (independiente de la hipótesis). Es conocida como probabilidad previa.

4.2.2.5. K vecinos más cercanos

El clasificador K vecinos más cercanos (KNN por sus siglas en inglés) es un algoritmo basado en instancias de clasificación. Su funcionamiento consiste en que una vez otorgada una base de datos de entrenamiento con valores etiquetados, el algoritmo calcula la distancia entre el dato a clasificar y el resto de datos de entrenamiento, para que luego cuando se le entrega un valor “ k ” de elementos mas cercanos (En Python como “`k_range`”), el algoritmo los busca dentro del hiperplano y en base a la etiqueta que más se repita se obtiene el resultado final (14).

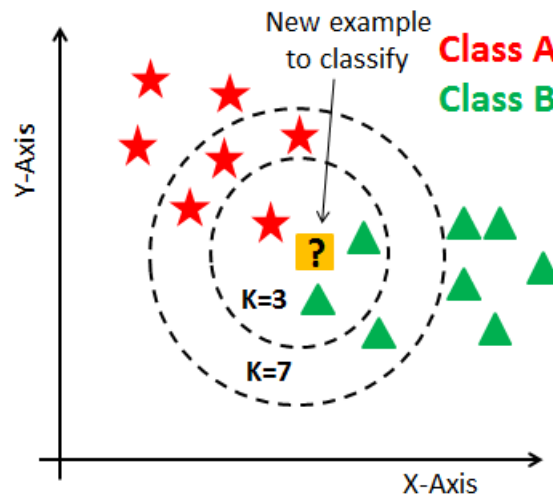


Figura 4.8: Ejemplo de funcionamiento KNN. Fuente: datacamp.com

4.2.3. Métricas de evaluación

Al programar un algoritmo de clasificación en Machine Learning, es de gran importancia evaluar el rendimiento que presenta, para lo cual existen diversas métricas dependiendo del tipo de datos. Al momento de clasificar en dos variables (para ejemplificar, positivos y negativos) se tienen 4 tipos de valores, los “positivos verdaderos (PV)” y “negativos verdaderos (NV)”, que son aquellos valores que fueron catalogados de forma correcta por el algoritmo, los denominados “positivos falsos (PF)” que son aquellos catalogados como positivos pero deberían ser negativos, y los “negativos falsos (NF)” que corresponden a los catalogados como negativos, pero son positivos.

Para observar los valores obtenidos mediante el modelo de predicción, se genera la denominada “matriz de confusión”. Esta matriz es de tamaño $n \times n$, donde “n” es el número de etiquetas que contiene el algoritmo. En la Tabla 4.1 se ve una matriz de confusión para un modelo de clasificación binaria.

real / predicción	negativo	positivo
negativo	NV	PF
positivo	NF	PV

Tabla 4.1: Matriz de confusión

En la Tabla 4.1 se observa que la matriz de confusión compara los valores obtenidos. Con estos resultados se pueden obtener varias métricas de evaluación, las cuales son:

4.2.3.1. Precisión (Precision)

La métrica de precisión mide la calidad del modelo de Machine Learning, en el caso de dos variables corresponde a la razón entre los datos “verdaderos” de una categoría, y la suma de los datos “verdaderos” y “falsos” de la misma categoría (15). Por ejemplo, para evaluar la precisión de los datos catalogados como positivos se hace de la forma:

$$precision = \frac{PV}{PV + PF} \quad (4.4)$$

4.2.3.2. Exhaustividad (Recall)

La métrica de exhaustividad se caracteriza por informar sobre la cantidad de datos que el modelo de Machine Learning catalogó de forma correcta, en comparación al total de datos que deberían haber sido clasificados en el mismo grupo (15). Por ejemplo, para evaluar la exhaustividad de los datos catalogados

como positivos se hace de la forma:

$$exhaustividad = \frac{PV}{PV + NF} \quad (4.5)$$

4.2.3.3. Exactitud (Accuracy)

La métrica de exactitud mide el porcentaje total de datos que el modelo ha clasificado correctamente. No es conveniente utilizarla cuando la cantidad de datos no esta balanceada para las distintas categorías, ya que puede dar un porcentaje alto, pero acertando a pocos resultados en la categoría que menos se repite (15). Para el mismo ejemplo anterior de datos positivos y negativos, se calcula de la forma:

$$Exactitud = \frac{PV + NV}{PV + NV + PF + NF} \quad (4.6)$$

4.2.3.4. Valor F1 (F1-score)

El Valor F1 es la métrica de exactitud que esta encargada de combinar los resultados de las medidas de precisión y exhaustividad en un solo valor. Es de gran utilidad cuando la cantidad de datos no esta balanceada, además de ofrecer la opción de darle mayor importancia a una de las dos variables mediante la regulación del factor “ β ” (15). Este valor se calcula de forma genérica de la forma:

$$F_{\beta} = (1 + \beta^2) * \frac{precision * exhaustividad}{(\beta^2 * precision) + exhaustividad} \quad (4.7)$$

Por ende, si se quiere obtener el valor F1, basta reemplazar “ β ” por 1, obteniendo finalmente:

$$F_1 = 2 * \frac{precision * exhaustividad}{precision + exhaustividad} \quad (4.8)$$

4.2.4. Procesamiento del Lenguaje Natural

El Procesamiento del Lenguaje Natural (PLN) es la técnica encargada de que las computadoras logren procesar y entender la forma en que se comunican los humanos, con el fin de utilizar esta información. Es definido como un campo interdisciplinario, puesto que combina la lingüística computacional, ciencias cognitivas e inteligencia artificial (16).

4.2.4.1. Tf-idf (Term Frequency - Inverse Document Frequency)

La representación semántica Tf-idf permite la vectorización de un documento, identificando el peso de cada palabra en relación al diccionario que se utiliza. Se representa como una matriz de datos en la cual se tiene tantas columnas como términos, y tantas filas como documentos (17). Este modelo se compone de:

- TF (Frecuencia de aparición de término): Este factor calcula la capacidad de representación del término en un documento, a través de la frecuencia con la que aparece. Su fórmula es:

$$Tf(n) = \sum D(n) \quad (4.9)$$

La cual indica que corresponde a la frecuencia de aparición de un término (n) en un documento (D), por lo que es la suma de sus ocurrencias.

- IDF (Frecuencia inversa del documento): Corresponde al coeficiente que determina la capacidad discriminatoria del término, con respecto al universo de palabras. Se calcula de la forma:

$$IDF(n) = \frac{N}{DF(n)} + 1 \quad (4.10)$$

En donde N es el número total de documentos y DF es el número de documentos donde aparece el término n.

- Ponderación TF-IDF: Es el producto de ambos factores y su valor representa la importancia del término en cada documento. Se calcula de la forma:

$$TF - IDF(n) = TF(n) * IDF(n) \quad (4.11)$$

4.3. Python

Python es un lenguaje de programación creado en 1991 de código abierto que está orientado a objetos de alto nivel y con semántica dinámica. Se caracteriza por la legibilidad del código, facilitando el orden y la productividad, motivo por el cual es utilizado para “scripting” y desarrollo de aplicaciones en diversas áreas, como lo son la ciencia de datos y la Inteligencia Artificial. Si bien fue creado como lenguaje de programación de uso general, a lo largo del tiempo se han creado diversas librerías, haciendo más fácil su uso al público general y aumentando su horizonte de alcance (18).

4.3.1. Jupyter Notebook

Jupyter Notebook es una aplicación y entorno de programación lanzado en 2015 por la organización sin fines de lucro Proyecto Jupyter (19). Permite, en diversos lenguajes de programación (entre ellos Python) la creación de documentos web en formato “.JSON” que representan un formato de lista ordenado de celdas de entrada (“In”) y salida (“out”), facilitando la comprensión del código programado.

4.3.2. Librerías utilizadas

1. **Pandas:** Es la librería encargada de permitir la importación y visualización de bases de datos en diversos formatos, como por ejemplo xlsx, csv, json, html, entre otros a Python. Es de gran utilidad para el tratamiento de datos, ya que permite cargarlos, modelarlos, analizarlos, manipularlos y prepararlos para su posterior uso en diversas aplicaciones, entre ellas el Aprendizaje Automático.
2. **Numpy:** Numpy es una librería de Python especializada en el cálculo numérico y el análisis de grandes volúmenes de datos. Dentro de esta librería se incluyen los arreglos (más conocidos como arrays en inglés) los cuales representan listas de forma eficiente, facilitando el tratamiento de vectores y matrices de grandes dimensiones (20).
3. **Matplotlib:** La librería matplotlib permite la generación de gráficos en dos dimensiones a partir de los datos entregados.
4. **Nltk:** Es una de las librerías de código abierto más utilizadas en Python para trabajar con datos de lenguaje humano y procesamiento del lenguaje natural, otorgando herramientas para clasificación, tokenización, limpieza de datos, etiquetado y razonamiento semántico (21). Esta librería posee diversos paquetes que facilitan el tratamiento de datos, los que son de utilidad para el trabajo de título son:

- **stem.porter:** Este paquete está encargado de reducir las palabras de la base de datos a su “stem” o

“raíz”. Al utilizarlo a las palabras se les elimina los caracteres que indiquen género, tiempo verbal o pluralismo. El objetivo de esto es reducir el diccionario total de palabras de la base de datos.

- stopwords: Las “stop words” son aquellas denominadas “palabras vacías” que por si solas carecen de sentido. En general los programas computacionales las ignoran, puesto que no entregan mayor información sobre el contexto de una frase. Estas palabras son básicamente conjunciones, artículos, preposiciones y adverbios. El paquete de NLTK “stopwords” viene cargado con un gran conjunto de palabras vacías, lo cual facilita la limpieza de los datos (22).

5. **Scikit-learn:** Scikit-learn es una librería utilizada para Machine Learning en Python, la cual contiene una gran cantidad de funciones. Los paquetes de interés para el presente trabajo son:

- CountVectorizer: Este paquete construye un vocabulario de palabras en base a los datos que se tengan. Al utilizarlo, lo que se entrega es un vector codificado con la longitud total del vocabulario y el número de veces que cada palabra se repite en la base de datos.
- TfidfTransformer: Es el paquete encargado de otorgarle el peso a cada palabra en base a su frecuencia de repetición dentro de la base de datos.
- train_test_split: Con este paquete se divide la cantidad total de datos en un conjunto de entrenamiento del algoritmo y otro de pruebas de precisión. La cantidad de datos para cada conjunto se entrega al momento de programar el código.
- svm: Librería para utilizar el modelo de aprendizaje supervisado de Máquina de soporte vectorial.
- DecisionTreeClassifier: Librería para utilizar el modelo de aprendizaje supervisado de Árboles de decisión.
- RandomForestClassifier: Librería para utilizar el modelo de aprendizaje supervisado de Bosques aleatorios.
- GaussianNB: Librería para utilizar el modelo de aprendizaje supervisado de Naïve Bayes.
- metrics: Esta librería se utiliza para obtener las métricas de un modelo de Machine Learning, como la matriz de confusión, precisión y exhaustividad, exactitud y Valor F1.

5 | Metodología

Para comenzar con el trabajo, en primer lugar se deben descargar los avisos generados por operadores, mantenedores y supervisores de la planta desde SAP PM, los cuales se utilizan para la programación del algoritmo.

5.1. Visualización de avisos

Dentro de SAP PM se tienen los avisos, a los cuales se puede acceder desde la transacción “iw29” (Figura 5.1).



Figura 5.1: Acceso a transacción iw29.

Dentro de la transacción, se completan los campos (Figura 5.2) ubicación técnica y rango de fe-

chas.

Figura 5.2: Campos para completar.

Una vez completados los campos, se visualizan los avisos de acuerdo a los filtros establecidos (Figura 5.3), los cuales además pueden ser descargados en formato “.xls”.

Prioridad	Creado el	Aviso	Status sistema	Descripción	Cl.	Duración de parada	Status de usuario	Orden	Ubicación técnica
2: Correctivo	14.06.2021	24139951	MECE ORAS	Instalar Anphoterol	M1	0,00	APRO	104839559	FFUN-PAS
1: Inmediato	16.06.2021	24144472	MECE	Planta de Acido		0,00	NUEV		FFUN-PAS
3: Planificado	08.06.2021	24125894	MECE ORAS	RUTA 136 LUB BBA PAS 1M	M4	0,00	NUEV	104826502	FFUN-PAS
3: Planificado	10.06.2021	24132826	MECE ORAS	INSP MON 1 S RUTA 137 .PAS OP		0,00	NUEV	104832749	FFUN-PAS
3: Planificado	18.06.2021	24150247	MECE ORAS	INSP MON 1 S RUTA 137 .PAS OP		0,00	NUEV	104849577	FFUN-PAS
3: Planificado	18.06.2021	24150258	MECE ORAS	INSP MON 2 S RUTA 161 PAS 25-OP		0,00	NUEV	104849589	FFUN-PAS
3: Planificado	25.06.2021	24168856	MECE ORAS	INSP MON 1 S RUTA 137 .PAS OP		0,00	NUEV	104867341	FFUN-PAS
						0,00			

Figura 5.3: Visualización de avisos.

5.2. Revisión fallas

5.2.1. Revisión frecuencia de fallas

Para comenzar se crea un documento en blanco de Jupyter Notebook, el cual se programa en el lenguaje Python y se le carga una planilla Excel obtenida del software SAP, con todas las fallas no planificadas reportadas en el período entre 02-01-2021 y 30-06-2021 (Anexo 1). Esta planilla se transforma en DataFrame, al que se le realiza una limpieza de las columnas que no son necesarias para el primer análisis, y posteriormente mediante el comando “str.contains(“AOAS-LAR”)” (Figura 5.4), se obtienen todas las fallas reportadas en los equipos dentro de la ubicación técnica establecida para el proceso de filtrado Larox.

```
In [4]: npa = np2[np2["Ubicación técnica"].str.contains("AOAS-LAR")]
npa.head()
```

```
Out[4]:
```

	Texto breve	Ubicación técnica
575	S#05 SIEMENS 7X7 reposion de raspador c#	AOAS-LAR
576	S#02 Siemens Reparacion de 02 Vat Plate	AOAS-LAR
577	S#15 SIEMENS 7X7 Traslado de materiales	AOAS-LAR
578	S#15 SIEMENS 7X7 Orden repuestos área La	AOAS-LAR
579	S#14 SIEMENS 7X7 Se traslada carro alzh	AOAS-LAR

```
In [6]: npa.shape
```

```
Out[6]: (333, 2)
```

Figura 5.4: Limpieza de base de datos y cantidad de fallas.

Luego, mediante el comando “shape” (Figura 5.4), se obtiene la cantidad de fallas reportadas en esta ubicación técnica, que en el caso de estudio sería igual a 333, la cantidad de filas del DataFrame.

Para facilitar la comprensión del análisis se crea un diccionario en Python, el cual transforma la ubicación técnica abreviada a su nombre completo (Figura 5.5).

```
In [10]: ut = ut[["Ubicación técnica", "Denominación de la ubicación técnica"]]
ut.head()
```

```
Out[10]:
```

	Ubicación técnica	Denominación de la ubicación técnica
0	AOAS	PLANTAS DE MOLIBDENO Y FILTRADO COBRE
1	AOAS-EFL	PLANTA EFLUENTES
2	AOAS-EFL-BEF	BOMBAS PLANTA EFLUENTES
3	AOAS-EFL-BEF-B01	BBA N°1 EFLUENTE PEROXIDO/BBA POZO DECAN
4	AOAS-EFL-BEF-B01-M01	MOTOR ELECTRICO

Figura 5.5: Ubicación técnica abreviada y su denominación.

Finalmente, en el código escrito se le asigna un valor de 1 a cada fila, para posteriormente mediante el comando “value_counts”, sumar las ubicaciones técnicas repetidas y obtener la cantidad de fallas reportadas para cada equipo de la planta en el período estudiado (Figura 5.6). Además, se grafican las 15 ubicaciones con más fallas dentro del proceso de filtrado Larox, como puede verse en la Figura 5.7.

```
In [14]: np = np.drop("Ubicación técnica", axis=1)
np["Cuenta"] = 1
np

...

In [20]: np["Ubicación"].value_counts()

Out[20]: FILTRO LAROX                                40
BOMBA N° 5 ESPESADOR CONCENTRADO N°2                28
BOMBA N° 1 ESPESADOR CONCENTRADO N°1                28
BOMBA N° 6 ESPESADOR CONCENTRADO N°2                20
BOMBA N° 3 ESPESADOR CONCENTRADO N°1                17
SISTEMA DE INSTRUMENTACIÓN FILTRO LAROX              1
BOMBA DE ENGRANAJES FILTRO LAROX                    1
SISTEMA DE INSTRUMENTOS Y CONTROL ELECT. ESPESADOR N° 2 1
CORREA DE TRASPASO 11 D                             1
VÁLVULA V17 FILTRO LAROX                            1
Name: Ubicación, Length: 101, dtype: int64
```

Figura 5.6: Cantidad de fallas para cada ubicación.

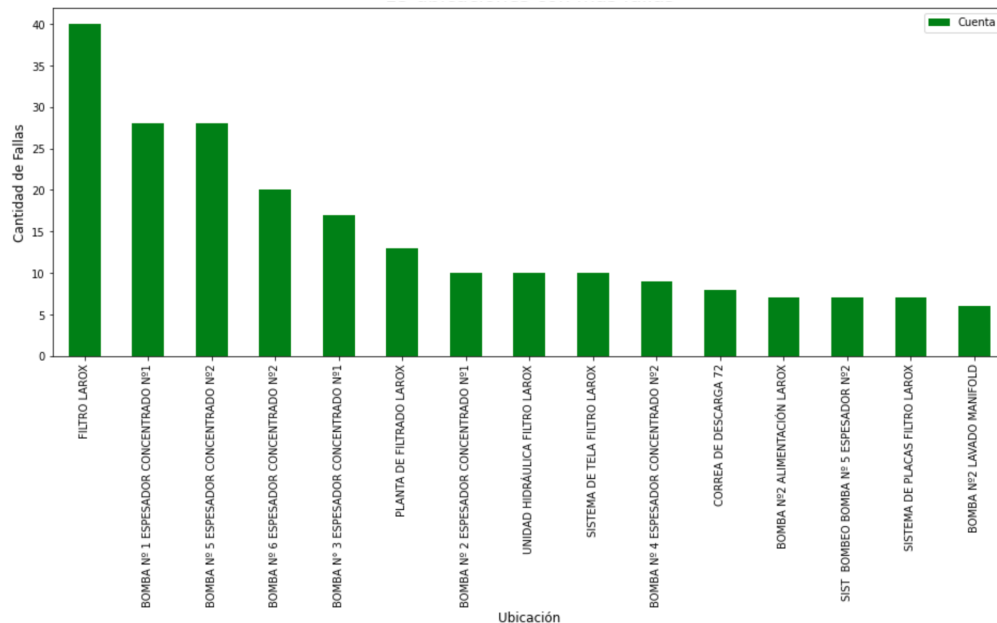


Figura 5.7: 15 ubicaciones técnicas con más fallas reportadas.

5.2.2. Análisis de planes de mantenimiento actuales

Al obtener un orden de las fallas no planificadas, se procede a hacer un análisis de Mantenimiento Centrado en Confiabilidad de cada falla (Anexo III), para luego revisar los actuales planes de mantenimiento que se poseen para los distintos equipos dentro de la planta de filtrado Larox, con el fin de revisar cual de estas fallas no tiene algún plan asociado, de donde se obtienen los siguientes resultados para los equipos relevantes del proceso:

■ Filtro Larox:

1. Paquete de placas: Se debe crear plan de mantenimiento, el cual debe incluir cambio de paquete de placas con frecuencia de cada un año, y complementar con plan de inspección mensual.
2. Sistema tensor de tela: Se debe crear plan de mantenimiento general con frecuencia de 6 meses.
3. Rodamientos batea: Se debe crear plan de mantenimiento general con frecuencia de 6 meses.
4. Válvulas V-1003 y V-1004: Se debe crear plan de mantenimiento mensual.
5. Dializadora de aceite: Se debe crear posición en SAP de su revisión dentro del plan semanal.

■ Bombas:

1. Agregar inspección de embancamiento cada 2 semanas.

■ Estanques:

1. Se debe agregar en el plan del agitador la instalación de lanzas.

■ Espesadores:

1. Sensor pH: Crear plan de cambio y revisión.
2. Cono: Crear plan de cambio.

■ Harnero:

1. Motor: Crear plan de revisión de VDF mensual.
2. Rodamiento: Incluir revisión en planes de MONCON (Monitoreo de condicion).

Luego de revisar los planes, los modos de falla reportados en el período de estudio se ordenan en una planilla Excel para los distintos equipos de la planta, con el fin de tener un consolidado de los mantenimientos que se han realizado, en conjunto a la frecuencia con que son reportados.

- **Filtro Larox:** Para este filtro se pueden ver 11 modos de falla distintos (Tabla 5.1), donde se puede destacar que la mayor cantidad de fallas reportadas tiene relación con válvulas y rotura de tela.

Tabla 5.1: Fallas reportadas en Filtro Larox

Equipo	Modo de falla	¿Tiene plan?	Cantidad de veces que se reporta
Filtro Larox	Falla de válvula	SI	20
Filtro Larox	Rotura de tela	SI	9
Filtro Larox	Paquete de placas desalineado	SI	4
Filtro Larox	Fuga en Vat Plate	SI	4
Filtro Larox	Falta de lubricación	SI	3
Filtro Larox	Falta de tensión de tela	NO	2
Filtro Larox	Cambio de junta	SI	2
Filtro Larox	Fuga de aceite	SI	2
Filtro Larox	Fisura de bomba	NO	2
Filtro Larox	Recirculación interna	NO	1

- **Bombas:** En las distintas bombas que son utilizadas en el proceso de Filtrado Larox, se puede ver que la falla más común es la de empaquetadura, mientras que el embancamiento de igual manera posee una alta frecuencia de repetición, y no tiene un plan asociado (Tabla 5.2).

Tabla 5.2: Fallas reportadas en bombas

Equipo	Modo de falla	¿Tiene plan?	Cantidad de veces que se reporta
Bombas	Cambio de empaquetadura	SI	20
Bombas	Bomba embancada	NO	9
Bombas	Falla de correas	SI	6
Bombas	Rotura línea de descarga	NO	5
Bombas	Cambio de válvula	SI	2
Bombas	Cambio de camisa	NO	2
Bombas	Desalineamiento	SI	1

- **Cintas:** Para las cintas transportadoras, las fallas que más se repiten son con respecto a los polines, y todas poseen un plan de mantenimiento asociado (Tabla 5.3).

Tabla 5.3: Fallas reportadas en cintas transportadoras

Equipo	Modo de falla	¿Tiene plan?	Cantidad de veces que se reporta
Cintas transp.	Retirar o reponer estacion de polines	SI	6
Cintas transp.	Correa descentrada	SI	4
Cintas transp.	Reposición estructuras, laterales y raspadores	SI	1

- **Espesadores de Concentrado:** En los espesadores de concentrado falta crear plan para el cambio de cono, evitando así su rotura (Tabla 5.4).
- **Harnero Delkor:** Para el Harnero Delkor las únicas fallas reportadas son con respecto a la falla de rodamientos, por lo que se debe incluir una inspección de monitoreo de condiciones, con el fin de poder predecir su desgaste (Tabla 5.5).

Tabla 5.4: Fallas reportadas en Espesadores de Concentrado

Equipo	Modo de falla	¿Tiene plan?	Cantidad de veces que se reporta
Espesadores de conc.	Cambio de aceite	SI	2
Espesadores de conc.	Rotura de cono	NO	1

Equipo	Modo de falla	¿Tiene plan?	Cantidad de veces que se reporta
Harnero Delkor	Falla de rodamientos	SI	3

Tabla 5.5: Fallas reportadas en Harnero Delkor

5.3. Revisión de avisos

Para el correcto funcionamiento del algoritmo de Machine Learning es de gran importancia la calidad de la base de datos de entrenamiento, debido a esto es que se hace una revisión de los avisos generados en el sistema SAP en la planta de filtrado Larox durante el período de enero - junio 2021.

5.3.1. Categorización de avisos de entrenamiento

De la base de datos inicial que será utilizada para el entrenamiento del algoritmo, en primer lugar se extraen todos los modos de falla reportados, para posteriormente realizar la evaluación de criticidad establecida en el SOMA a cada uno de ellos, y así evaluar si un posterior aviso a predecir será catalogado como de importancia Leve (Nota 1), Moderada (Nota 2) o Alta (Nota 3). Esta evaluación puede verse en el Anexo 1.

Luego de categorizar la importancia de los modos de falla reportados, se procede a crear una nueva columna en el Excel obtenido de SAP, en la cual se escribe el nivel otorgado (1, 2 o 3) (Figura 5.8). El objetivo de agregar esta información a la base de datos es entrenar el algoritmo de Machine Learning con datos reales.

Ubicación técnica	Descripción	nivel
AOAS-LAR-FLA	Cambiar Tela Filtrante	2
AOAS-LAR	FABRICACION GREATING CA	3
AOAS-LAR-FLA-MHD	Cambiar Juanta Elastica Motor HM8 Larox	2
AOAS-LAR-FLA-HID	Rellenar aceite filtro larox	3
AOAS-LAR-CTR-C01	centrado correa 72	1
AOAS-LAR-FLA	aramado de catrich	3
AOAS-LAR-FLA-VAL-VCN-V01	LAROX;FILTRO SE DETIENE POR FALLA V/V V3	3
AOAS-LAR-FLA	REPARACION VAT PLATE.	3
AOAS-LAR-ETQ-E06	ALROX: AIRE SECAD ACUMULADOR NO DESPICHA	1
AOAS-LAR-FLA-PLA	Reparar vatplate Sector Taller	3
AOAS-LAR	Reparacion de 02 Vat Plate	3
AOAS-LAR-FLA-PLA	Reparacion de vat-plate	3
AOAS-LAR-CTR-C01	Reponer estaciones de Polines correa 72	2

Figura 5.8: Extracto de base de datos de entrenamiento.

5.3.2. Programación de algoritmo de pruebas

Una vez finalizada la creación de la base de datos de entrenamiento, se programa el algoritmo en Python, en el cual en primer lugar se realiza un tratamiento de la información, para posteriormente probar con distintas técnicas de Machine Learning y evaluar entre ellas cual obtiene un mayor porcentaje de precisión. Para comenzar se carga la base de datos bajo la variable “data”, para luego mediante los comandos “shape” y “value_counts” obtener información sobre el tamaño de la data. Como puede verse en la Figura 5.9, se tiene una base de datos con 427 valores, en donde se han categorizado 223 con nota 3, 110 con nota 2 y 94 con nota 1.

```
In [5]: data = data[["Descripción", "nivel"]]
data.head()

Out[5]:
```

	Descripción	nivel
0	Cambiar Tela Filtrante	2
1	FABRICACION GREATING CA	3
2	Cambiar Juanta Elastica Motor HM8 Larox	2
3	Rellenar aceite filtro larox	3
4	centrado correa 72	1

```
In [6]: data.shape

Out[6]: (427, 2)

In [7]: data['nivel'].value_counts()

Out[7]: 3    223
        2    110
        1     94
        Name: nivel, dtype: int64
```

Figura 5.9: Información del tamaño base de datos de entrenamiento.

5.3.2.1. Limpieza de base de datos

En la programación del algoritmo de pruebas, uno de los primeros pasos es realizar una limpieza de la base de datos, por lo que en el caso en estudio se definen dos variables, “X” que es la columna “Descripción” de la base de datos de entrenamiento, mientras que “y” es la columna “Nivel” otorgado a cada falla (Figura 5.10).

```

In [8]: X = data['Descripción']
        y = data["nivel"]

In [9]: X.shape
Out[9]: (427,)

In [10]: y.shape
Out[10]: (427,)

```

Figura 5.10: Definición de variables X e y.

5.3.2.2. Preprocesado de información

Una vez cargadas las variables a utilizar, es importante para el correcto funcionamiento del algoritmo tener la información más clara y resumida posible, por lo que uno de los pasos del preprocesado es eliminar los caracteres que no sean letras (por ejemplo los puntos, comas, signos de exclamación), además de también eliminar las denominadas “stopwords” o palabras vacías, que son aquellas que no tienen un significado especial como los pronombres, artículos, entre otros (Figura 5.11). La información preprocesada es guardada bajo la variable “X_limpio”.

```

In [11]: X_limpio = []
        for i in X:
            for carac in ['.', ',', ':', ';', '!', '?', '|', '@']:
                i=i.lower().replace(carac, "")
            lineanueva=""
            for pal in i.split():
                if pal not in stopwords:
                    lineanueva=lineanueva+stemmer.stem(pal)+" "
            X_limpio.append(lineanueva)
        #print (X_preprocessed)
        X_limpio=np.asarray(X_limpio)

```

Figura 5.11: Código para el preprocesamiento de la información.

Posterior a obtener el “X_limpio”, para el correcto funcionamiento del código de Machine Learning se debe convertir las palabras a vectores de números, para lo cual se utiliza la técnica de TF-IDF, obteniendo finalmente la variable “X_preprocessed”. Como puede verse en la Figura 5.12, cada aviso esta representado dentro de un vector de 534 valores.

```

In [13]: from sklearn.feature_extraction.text import CountVectorizer
         from sklearn.feature_extraction.text import TfidfTransformer
         from sklearn.pipeline import Pipeline

         preprocessing = Pipeline([
             ('vect', CountVectorizer()),
             ('tfidf', TfidfTransformer())
         ])

         print("Preprocesando data...")
         X_preprocessed = preprocessing.fit_transform(X_limpio)

         Preprocesando data...

In [14]: X_preprocessed.shape
Out[14]: (427, 534)

```

Figura 5.12: Vectorización TF-IDF.

5.3.2.3. División de datos para entrenamiento y prueba

Una vez se tienen las variables de entrada preprocesadas, se procede a dividirlos en datos de entrenamiento y de prueba (train y test, Figura 5.13). Los de entrenamiento son utilizados para el aprendizaje del algoritmo, mientras que los de prueba son evaluados por el código y comparados con su valor real, para así definir la precisión del algoritmo. En este caso, se utiliza un 20 % de los datos (86 datos) para el proceso de testing.

```

In [15]: from sklearn.model_selection import train_test_split

In [16]: X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y, test_size=0.2, random_state=1)

In [17]: X_train.shape
Out[17]: (341, 534)

In [18]: y_test.shape
Out[18]: (86,)

```

Figura 5.13: División de datos en entrenamiento y prueba.

5.3.2.4. Prueba de Máquina Soporte Vectorial

El primer método de Machine Learning que se evalúa es el Máquina Soporte Vectorial, por lo que para comenzar, se importan las librerías necesarias (Figura 5.14).

```
In [25]: from sklearn.svm import SVC
         from sklearn.metrics import f1_score

In [26]: modelSVC = SVC()
```

Figura 5.14: Importación de librerías de SVM.

Para mejorar aún más la precisión del algoritmo, se utiliza la librería “GridSearchCV”, la cual prueba distintas combinaciones de valores para encontrar los parámetros del código que más se ajustan al modelo (Figura 5.15).

```
In [33]: from sklearn.model_selection import GridSearchCV

In [34]: param_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001]}

In [35]: grid = GridSearchCV(SVC(), param_grid, scoring="f1_weighted", verbose = 2)

In [36]: grid.fit(X_train, y_train)

Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV] END .....C=0.1, gamma=1; total time= 0.0s
[CV] END .....C=0.1, gamma=1; total time= 0.0s
[CV] END .....C=0.1, gamma=1; total time= 0.0s
[CV] END .....C=0.1, gamma=1; total time= 0.0s
[CV] END .....C=0.1, gamma=1; total time= 0.0s
[CV] END .....C=0.1, gamma=0.1; total time= 0.0s
[CV] END .....C=0.1, gamma=0.1; total time= 0.0s
[CV] END .....C=0.1, gamma=0.1; total time= 0.0s
[CV] END .....C=0.1, gamma=0.1; total time= 0.0s
[CV] END .....C=0.1, gamma=0.1; total time= 0.0s
[CV] END .....C=0.1, gamma=0.01; total time= 0.0s
[CV] END .....C=0.1, gamma=0.01; total time= 0.0s
[CV] END .....C=0.1, gamma=0.01; total time= 0.0s
[CV] END .....C=0.1, gamma=0.01; total time= 0.0s
[CV] END .....C=0.1, gamma=0.001; total time= 0.0s
[CV] END .....C=0.1, gamma=0.001; total time= 0.0s
[CV] END .....C=0.1, gamma=0.001; total time= 0.0s
[CV] END .....C=0.1, gamma=0.001; total time= 0.0s
[CV] END .....C=0.1, gamma=0.001; total time= 0.0s
[CV] END .....C=0.1, gamma=0.001; total time= 0.0s

In [37]: grid.best_params_
Out[37]: {'C': 10, 'gamma': 0.1}
```

Figura 5.15: GridSearch para SVM.

Una vez obtenidos los parámetros que entregan un mejor resultado, se realiza la predicción de valores con los datos de prueba, en donde se obtiene una evaluación f1-score de un 75 % (Figura 5.16).

```

In [38]: grid_pred = grid.predict(X_test)

In [39]: print(confusion_matrix(y_test, grid_pred))
          print(classification_report(y_test, grid_pred))

[[10  0  8]
 [ 1 11  7]
 [ 4  1 44]]

```

	precision	recall	f1-score	support
1	0.67	0.56	0.61	18
2	0.92	0.58	0.71	19
3	0.75	0.90	0.81	49
accuracy			0.76	86
macro avg	0.78	0.68	0.71	86
weighted avg	0.77	0.76	0.75	86

```

In [40]: f1_score(y_test, grid_pred, average='weighted')

Out[40]: 0.7478940442181252

```

Figura 5.16: Predicción de SVM.

5.3.2.5. Prueba de Árboles de Decisión

El segundo modelo de Machine Learning que se prueba es el de Árboles de Decisión, para el cual en primer lugar se importa la librería requerida, y luego mediante la generación de un gráfico poder representar que profundidad de árboles presenta un “accuracy” más alto, probando con números entre 1 y 20 saltando de 0.5 en 0.5, obteniendo que el valor con una precisión más alta es con una profundidad de 11. (Figura 5.17).

```

In [30]: resultados=[]
depth_range=range(1,20)
for i in depth_range:
    dtree=DecisionTreeClassifier(max_depth=i)
    dtree.fit(X_train, y_train)
    dectre_pred = dtree.predict(X_test)
    resultados.append(metrics.accuracy_score(y_test, dectre_pred))
# import Matplotlib (scientific plotting library)
import matplotlib.pyplot as plt

# allow plots to appear within the notebook
%matplotlib inline

# plot the relationship between K and testing accuracy
plt.plot(depth_range, resultados)
plt.xlabel('Value of Depth for DT')
plt.ylabel('Testing Accuracy')

```

Out[30]: Text(0, 0.5, 'Testing Accuracy')

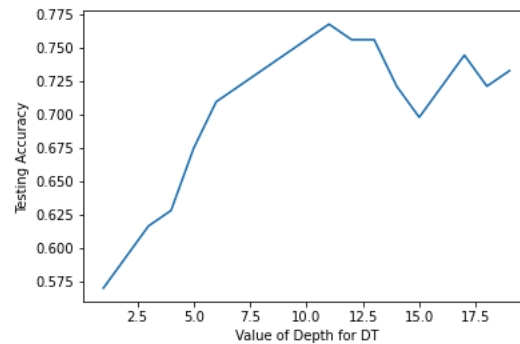


Figura 5.17: Testing Accuracy para Árboles de Decisión.

Una vez obtenido el parámetro que da el mejor resultado, se procede a definir el modelo, para posteriormente con los datos de prueba obtener el valor f1-score de un 74 % (Figura 5.18).

```

In [40]: dtree=DecisionTreeClassifier(max_depth=11)
dtree.fit(X_train,y_train)
dtree_pred = dtree.predict(X_test)

In [41]: print(confusion_matrix(y_test, dtree_pred))
print(classification_report(y_test, dtree_pred))

```

	[[6 0 12]				
	[0 13 6]				
	[0 2 47]]				
		precision	recall	f1-score	support
1	1.00	0.33	0.50	18	
2	0.87	0.68	0.76	19	
3	0.72	0.96	0.82	49	
accuracy			0.77	86	
macro avg	0.86	0.66	0.70	86	
weighted avg	0.81	0.77	0.74	86	

```

In [42]: f1_score(y_test, dtree_pred, average='weighted')

Out[42]: 0.743406052751578

```

Figura 5.18: Predicción de Árboles de Decisión.

5.3.2.6. Prueba de Bosques Aleatorios

El siguiente modelo de Machine Learning a evaluar es el de Bosques Aleatorios, por lo que se comienza por importar la librería respectiva, llamada “RandomForestClassifier”. Luego de esto, se debe elegir los parámetros con los que se realiza la predicción, para lo cual se utiliza la función “GridSearchCV” en donde se le entrega una cantidad de entre 60 y 200 “n_estimators” y los criterios “gini” y “entropy”, para que encuentre la combinación que entrega una mayor métrica f1-score (Figura 5.19).

```
In [50]: from sklearn.ensemble import RandomForestClassifier

In [51]: rfcl=RandomForestClassifier(random_state=33)

In [52]: param_grid = {
          'n_estimators': [60,200],
          'criterion': ['gini', 'entropy']
        }

In [53]: CV_rfc = GridSearchCV(estimator=rfcl, param_grid=param_grid, scoring="f1_weighted", cv= 5)
          CV_rfc.fit(X_train, y_train)

Out[53]: GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=33),
                    param_grid={'criterion': ['gini', 'entropy'],
                                'n_estimators': [60, 200]},
                    scoring='f1_weighted')

In [54]: CV_rfc.best_params_

Out[54]: {'criterion': 'gini', 'n_estimators': 200}
```

Figura 5.19: Librerías y GridSearch en Random Forest.

Como puede verse en la Figura 5.19, se obtiene un mejor resultado para “criterion” = “gini” y “n_estimators” = 200, por lo que se hace la predicción con estos valores, y se obtiene un f1 final de un 80 % (Figura 5.20).

```
In [62]: print(confusion_matrix(y_test, rfc_pred))
          print(classification_report(y_test, rfc_pred))

[[10  0  8]
 [ 0 13  6]
 [ 1  1 47]]
              precision    recall  f1-score   support

         1       0.91      0.56      0.69         18
         2       0.93      0.68      0.79         19
         3       0.77      0.96      0.85         49

   accuracy                   0.81         86
  macro avg       0.87      0.73      0.78         86
 weighted avg       0.83      0.81      0.81         86

In [63]: f1_score(y_test, rfc_pred, average='weighted')

Out[63]: 0.8053048528589827
```

Figura 5.20: Resultado de Random Forest.

5.3.2.7. Prueba de Naïve Bayes

Para evaluar el desempeño de Naïve Bayes, al igual que en los casos anteriores, se comienza importando la librería necesaria y llamando a la variable “nb” como el clasificador bayesiano (Figura 5.21).

```
In [67]: from sklearn.naive_bayes import GaussianNB
In [68]: nb = GaussianNB()
In [69]: nb.fit(X_train.toarray(), y_train)
Out[69]: GaussianNB()
```

Figura 5.21: Definición del algoritmo de Naïve Bayes.

Luego de definirse el clasificador bayesiano, se procede a realizar la predicción con los datos de pruebas. Puede verse en la matriz de confusión (Figura 5.22) que el valor de f1 es inferior a los anteriores, variando entre 47 % y 64 %.

```
In [70]: nb_pred = nb.predict(X_test.toarray())

In [71]: print(confusion_matrix(y_test,nb_pred))
print(classification_report(y_test,nb_pred))
```

```
[[11  3  4]
 [ 2 11  6]
 [ 7 14 28]]
```

	precision	recall	f1-score	support
1	0.55	0.61	0.58	18
2	0.39	0.58	0.47	19
3	0.74	0.57	0.64	49
accuracy			0.58	86
macro avg	0.56	0.59	0.56	86
weighted avg	0.62	0.58	0.59	86

Figura 5.22: Matriz de confusión algoritmo de Naïve Bayes.

5.3.2.8. Prueba de KNN

La última técnica de Machine Learning a evaluar es la de KNN, para la cual en primer lugar se importa la librería necesaria, para posteriormente evaluar el comportamiento del algoritmo mediante un ciclo “for”, el cual prueba con K vecinos entre 1 y 100 (Fig. 5.23).

```

In [72]: from sklearn.neighbors import KNeighborsClassifier

In [73]: k_range = range(1, 100)
         scores = []
         for k in k_range:
             knn = KNeighborsClassifier(n_neighbors=k)
             knn.fit(X_train, y_train)
             pred_knn = knn.predict(X_test)
             scores.append(metrics.accuracy_score(y_test, pred_knn))

```

Figura 5.23: Ciclo “for” para evaluar distintos valores de K.

Luego de definir el ciclo “for” se grafican los valores, como puede verse en la Figura 5.24. Si bien no hay una tendencia claramente definida, se puede observar que la precisión en general baja (Partiendo de un 70 % con 1 vecino, hasta llegar a un 58 % aproximadamente con 100 vecinos), por lo que se concluye que no es una buena metodología para evaluar los datos en estudio.

```

In [74]: # importar Matplotlib (librería de gráficos)
         import matplotlib.pyplot as plt

         # Permite que los gráficos sean visibles en el notebook
         %matplotlib inline

         # Grafica la relación entre K vecinos y Accuracy
         plt.plot(k_range, scores)
         plt.xlabel('Value of K for KNN')
         plt.ylabel('Testing Accuracy')

```

Out[74]: Text(0, 0.5, 'Testing Accuracy')

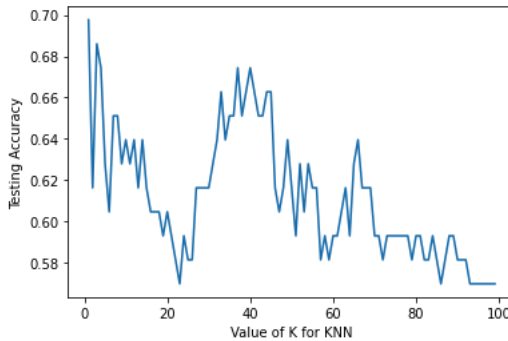


Figura 5.24: Gráfica de precisión con K vecinos.

5.3.3. Programación de algoritmo final

5.3.3.1. Carga de librerías y bases de datos

Realizadas las pruebas para las distintas técnicas de Machine Learning en estudio, la que obtiene un mejor resultado de métrica “f1” es la de Bosques Aleatorios (Tabla 5.6, por lo que es la escogida para la programación del algoritmo final.

Tabla 5.6: Resultado f1-score para distintas técnicas utilizadas

Técnica	f1-score
Maquina soporte vectorial	75 %
Árboles de decisión	74 %
Bosques aleatorios	80 %
Naïve Bayes	64 %
KNN	70 %

Para comenzar la programación del algoritmo, en primer lugar se importan todas las librerías necesarias para el correcto funcionamiento (Figura 5.25).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk
from nltk.stem.porter import *
import re
import six
nltk.download('rsdp')
nltk.download('stopwords')
pd.options.display.max_columns = None
pd.options.display.max_rows = None
from sklearn import metrics
```

Figura 5.25: Librerías requeridas para el algoritmo.

Luego se cargan las dos bases de datos que se utilizan, la llamada “train” son los datos de entrenamiento del algoritmo y siempre debe estar en la misma carpeta que el notebook de Python, y también se carga el archivo “data” (Figura 5.26), el cual es la base de datos que se desea categorizar y se debe ir actualizando a medida que se quiera hacer una nueva predicción.

A los datos nuevos ingresados se les hace una limpieza, eliminando todas las celdas que contengan palabras con “Mant”, “Mtto” y “Prep”, ya que son avisos que no contienen información relevante (Figura 5.26)

```
data_train = pd.read_excel("train.xlsx")

data = pd.read_excel("AVISOS.xlsx")

eliminar = ["MTTO", 'Mtto', 'mtto', 'Mant', 'MANT', 'mant', 'prep', 'PREP', 'Prep']
data = data[~data['Descripción'].str.contains('|'.join(eliminar))]

data = data.reset_index(level=None, drop=False, inplace=False, col_level=0, col_fill='')
```

Figura 5.26: Carga de bases de datos.

5.3.3.2. Generación de reporte de datos

Teniendo los archivos cargados y depurados, para ayudar al análisis e interpretación de los datos se hace un reporte por ubicación, el cual entrega dos imágenes que son descargadas al computador en que se hace correr el algoritmo, la primera, es un listado con las ubicaciones y la cantidad de avisos reportados para la cada una de ellas (Figura 5.27), mientras que la segunda, es un gráfico el cual muestra las 15 ubicaciones técnicas con más avisos (Figura 5.28)

Ubicación	Cantidad de avisos
SISTEMAS ELÉCTRICOS	3
PLANTA DE FILTRADO LAROX	2
BOMBA N° 4 ESPESADOR CONCENTRADO N°2	2
HARNERO DELKOR PLANTA LAROX	2
ESPESADOR DE CONCENTRADO N°2	1
SIST BOMBEO BOMBA N°2 LAVADO TELA	1
MOTOR ELE SIST TRANSMISION CORREA 72	1

Figura 5.27: Fragmento de imagen de ubicaciones y avisos entregado por el algoritmo.

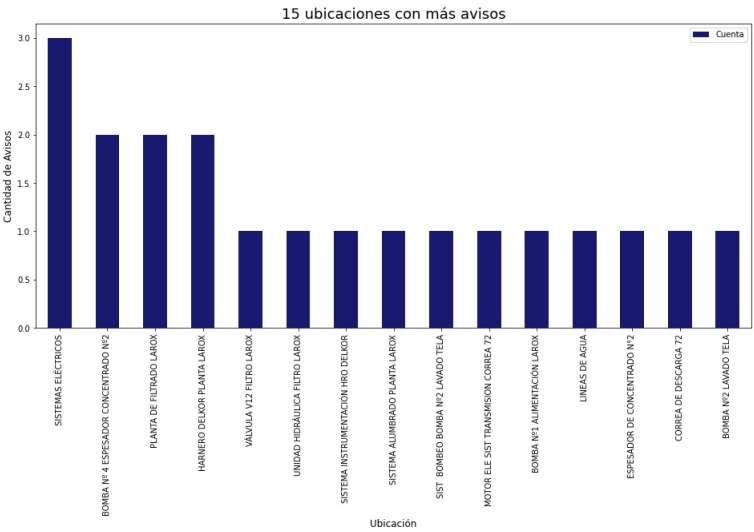


Figura 5.28: Imagen entregada por el algoritmo con ubicaciones técnicas con más avisos.

5.3.3.3. Programación de algoritmo de predicción

Para programar el algoritmo de predicción, el primer paso es definir las variables que se utilizarán (Figura 5.29), las cuales son:

- X = La variable “X” es la columna “Descripción” de la base de datos de entrenamiento, que corresponde al texto de cada aviso.
- y = La variable “y” es la columna “nivel”, que es el valor asignado a cada dato de la columna “Descripción”.
- X_pred = La variable “X_pred” son los valores de la columna “Descripción” de la base de datos nueva ingresada, que corresponden a aquellos que se les predice su nivel de importancia.

```
X = data_train['Descripción']
y = data_train["nivel"]
X_pred = data["Descripción"]
```

Figura 5.29: Definición de variables.

Luego de definir las variables, al igual que en la fase de pruebas de algoritmos es necesario preprocesar los datos que corresponden a texto, a los cuales nuevamente se les eliminan los caracteres que correspondan a símbolos y no sean letras, además de eliminar las “stopwords”, con el fin de obtener la información de la forma más precisa posible. La variable “X” depurada se guarda bajo el nombre de “X_limpio”, mientras que la variable “X_pred” luego de ser limpiada se guarda como “X_predvec” (Figura 5.30).

```
X_limpio = []
for i in X:
    for carac in ['.', ',', ':', ';', '!', '?', '|', '@']:
        i=i.lower().replace(carac, "")
    lineanueva=""
    for pal in i.split():
        if pal not in stopwords:
            lineanueva=lineanueva+stemmer.stem(pal)+" "
    X_limpio.append(lineanueva)
#print (X_preprocessed)
X_limpio=np.asarray(X_limpio)

X_predvec = []
for i in X_pred:
    for carac in ['.', ',', ':', ';', '!', '?', '|', '@']:
        i=i.lower().replace(carac, "")
    lineanueva=""
    for pal in i.split():
        if pal not in stopwords:
            lineanueva=lineanueva+stemmer.stem(pal)+" "
    X_predvec.append(lineanueva)
#print (X_preprocessed)
X_predvec=np.asarray(X_predvec)
```

Figura 5.30: Limpieza de variables X.

El siguiente paso es convertir las palabras a vectores de números, por lo que mediante la vectorización TF-IDF se transforman las variables “X_limpio” y “X_predvec” en “X_preprocessed” y “Xpred_preprocessed” respectivamente (Figura 5.31).

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.pipeline import Pipeline

preprocessing = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer())
])

print("Preprocesando data...")
X_preprocessed = preprocessing.fit_transform(X_limpio)

print("Preprocessing test data...")
Xpred_preprocessed = preprocessing.transform(X_predvec)

Preprocesando data...
Preprocessing test data...

X_preprocessed.shape
(427, 534)

Xpred_preprocessed.shape
(22, 534)
```

Figura 5.31: Vectorización de variables.

Como puede verse en la Figura 5.31, mediante el comando “shape” se obtiene que se tienen 427 datos de entrenamiento y se predecirán 22 datos, formándose así un espacio vectorial de 534 palabras, representadas como números.

Con los datos ya preprocesados y vectorizados se puede programar la sección de Machine Learning del algoritmo, utilizando los valores obtenidos en la fase de entrenamiento de “n_estimators” = 200 y “criterion” = “gini” (Figura 5.32)

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators = 200, criterion = 'gini', random_state = 33)
```

Figura 5.32: Programación de Bosques Aleatorios.

Luego de definir el algoritmo de clasificación, se procede a hacer la predicción del conjunto de datos “Xpred_preprocessed”, este se guarda en la variable “rfc_pred” mediante la utilización del comando “rfc.predict(X_preprocessed)”. Posteriormente, en el DataFrame de nombre “data” se definen 3 columnas las cuales son “Ubicación”, “Descripción” y se define una nueva llamada “Nivel de importancia”. Esta última, mediante el comando “ascending=False” se ordena de mayor a menor, para así dejar aquellas con clasificación 3 (La más alta) en el primer lugar, ordenando los avisos de mayor a menor importancia (5.33).

```

In [23]: rfc_pred = rfc.predict(Xpred_preprocessed)

In [24]: data = data.loc[:,['Ubicación', 'Descripción']]
data = data[['Ubicación', 'Descripción']]

In [25]: DFpred = pd.DataFrame(rfc_pred)
DFpred.reset_index(drop = True, inplace = True)
df_pred = data.join(DFpred)

In [26]: df_pred = df_pred.rename(columns={0: 'Nivel de importancia'})

In [27]: df_pred = df_pred.sort_values('Nivel de importancia', ascending=False)

```

Figura 5.33: Programación de DataFrame “data”.

Para facilitar la comprensión del archivo final, con el comando “replace” se cambian los valores 3, 2 y 1 por “ALTA”, “MEDIA” y “BAJA” respectivamente, refiriéndose al nivel de importancia representado. Además, el resultado final se guarda en la carpeta donde se guarda el código bajo el nombre “Predicción de Avisos.xlsx” (Figura 5.34).

```

In [29]: df_pred.to_excel('Predicción de Avisos.xlsx', index = False)

In [30]: df_pred.head()

Out[30]:

```

	Ubicación	Descripción	Nivel de importancia
11	BOMBA N° 4 ESPESADOR CONCENTRADO N°2	DÚcto de alimentacion con daño	ALTA
2	SISTEMA INSTRUMENTACIÓN HRO DELKOR	Instalacion V/V ARNERO DELKOR	ALTA
3	PLANTA DE FILTRADO LAROX	Cambio de Correa 11D	ALTA
4	VÁLVULAS FILTRO LAROX	CAMBIO V/V V2-V12	ALTA
5	SISTEMAS ELÉCTRICOS	Trabaos Habilitacion U/H LAROX	ALTA

Figura 5.34: Obtención de DataFrame final.

Finalmente para mejorar la obtención y visualización de resultados se define la función “render2_mpl_table”, la cual transforma el DataFrame obtenido en una imagen, que es guardada en la carpeta donde se guarda el código con el nombre de “Predicción.jpg” (Figura 5.35).

```
In [31]: def render2_mpl_table(data, col_width=3.0, row_height=0.625, font_size=14,
        header_color='#40466e', row_colors=['#f1f1f2', 'w'], Edge_color='w',
        bbox=[0, 0, 1, 1], header_columns=0,
        ax=None, **kwargs):
    if ax is None:
        size = (np.array(data.shape[:-1]) + np.array([0, 1])) * np.array([col_width, row_height])
        fig, ax = plt.subplots(figsize=size)
        ax.axis('off')

    mpl_table = ax.table(cellText=data.values, bbox=bbox, colLabels=data.columns, **kwargs)

    mpl_table.auto_set_font_size(False)
    mpl_table.set_fontsize(font_size)

    for k, cell in six.iteritems(mpl_table._cells):
        cell.set_edgecolor(Edge_color)
        if k[0] == 0 or k[1] < header_columns:
            cell.set_text_props(weight='bold', color='w')
            cell.set_facecolor(header_color)
        else:
            cell.set_facecolor(row_colors[k[0]%len(row_colors) ])
    return fig.savefig('Prediccion.jpg')

render2_mpl_table(df_pred, header_columns=0, col_width=7.0)
```

Ubicación	Descripción	Nivel de importancia
BOMBA Nº 4 ESPESADOR CONCENTRADO Nº2	Ducto de alimentacion con daño	ALTA
SISTEMA INSTRUMENTACIÓN HRO DELKOR	Instalacion V/V ARNERO DELKOR	ALTA
PLANTA DE FILTRADO LAROX	Cambio de Correa 11D	ALTA

Figura 5.35: Obtención de imagen final.

5.4. Análisis de resultados

Al obtener los resultados del algoritmo de predicción, se hace la evaluación de cada aviso con el estudio de RCM inicial con el fin de verificar si la categorización fue correcta o no. Para facilitar la obtención del porcentaje de precisión, se le asigna el valor “1” a la predicción bien hecha, y el valor “0” a la que presenta error (Figura 5.36).

Ubicación	Descripción	Nivel de importancia	Bien (1) / Mal (0)
BOMBA Nº 4 ESPESADOR CONCENTRADO Nº2	ROTURA DE LINEA	ALTA	1
SIST BOMBEO BOMBA Nº2 LAVADO TELA	bomba 2 con fuga	ALTA	0
BOMBA Nº2 ALIMENTACIÓN LAROX	mantencion a bba numero 2 alim. larox	ALTA	0
BOMBA Nº1 ALIMENTACIÓN LAROX	Bba 1 alimentación larox baja eficiencia	ALTA	1
BOMBA Nº 4 ESPESADOR CONCENTRADO Nº2	DUcto de alimentacion con daño	ALTA	1
PLANTA DE FILTRADO LAROX	CAMB.ELEM.ELÁST.ACOPLE.MOTOR.LÁROX	ALTA	1
SISTEMAS ELÉCTRICOS	S#43 SIEMENS Instalar iluminación correa	ALTA	0
UNIDAD HIDRÁULICA FILTRO LAROX	TENDIDO MANG.HIDRAULICAS NUEVA UNIDAD.	ALTA	1
FILTRO LAROX	PREPARATIVOS MANTENCION MAYOR	ALTA	0
VÁLVULA V12 FILTRO LAROX	válvula pasandose	ALTA	1
HARNERO DELKOR PLANTA LAROX	Línea drenaje harnero delkor embancada	ALTA	1
LINEAS DE AGUA	Repara linea alim. agua tk lav. manifold	ALTA	1
SISTEMAS ELÉCTRICOS	Trabaos Habilitacion U/H LAROX	ALTA	1
VÁLVULAS FILTRO LAROX	CAMBIO V/V V2-V12	ALTA	1
PLANTA DE FILTRADO LAROX	Cambio de Correa 11D	ALTA	1
SISTEMA INSTRUMENTACIÓN HRO DELKOR	Instalacion V/V ARNERO DELKOR	ALTA	1
HARNERO DELKOR PLANTA LAROX	Cambio de rodillo	MEDIA	0
SISTEMA ALUMBRADO PLANTA LAROX	Inst o Cambiar Focos sobre Bbas Manifold	MEDIA	1
ESPESADOR DE CONCENTRADO Nº2	Cambio De Rastras Espesaador #2	MEDIA	1
BOMBA Nº2 LAVADO MANIFOLD	Cambio bombas Grundfos CR-45-4	MEDIA	1
BOMBA Nº2 LAVADO TELA	Bomba 2 lavado de tela con fuga	MEDIA	1
LINEAS DE ALIMENTACIÓN FILTRO LAROX	PREPARATIVO CAMBIO TRAMO CAÑERÍA BBA 4	MEDIA	1
CORREA DE DESCARGA 72	Cambio de Polines correa 72	MEDIA	1
SISTEMAS ELÉCTRICOS	INSUMOS TRABAJOS FILTRO LAROX	BAJA	1
FILTRO LAROX	MTTO.MECANICO FILTRO LAROX	BAJA	1
FILTRO LAROX	MTTO.MECANICO FILTRO LAROX	BAJA	1
MOTOR ELE SIST TRANSMISION CORREA 72	Limpiar Motor Correa 72 Mucho Concentrad	BAJA	1
			81%

Figura 5.36: Tabla comparativa.

Como puede verse en la Figura 5.36, de los 27 avisos que se le hizo una predicción, en 22 de ellos la predicción fue correcta, mientras que en sólo 5 de ellos hubo un error, por lo que se obtiene que el porcentaje de precisión es de un 81 %, superior al 80 % que se obtuvo en la fase de pruebas de algoritmos.

6 | Conclusiones

Como se pudo ver en el presente trabajo, la política de producción y mantenimiento de Codelco velan por sostener constantemente la continuidad operacional y aumentar la vida útil de los activos. Es fundamental para estos objetivos la periódica investigación y desarrollo de tecnologías que aporten en la predicción de fallas y automatización de tareas, por lo que la programación e implementación de diferentes técnicas ligadas a la Inteligencia Artificial presentan una propuesta de valor a la gestión del mantenimiento.

En primer lugar, mediante un estudio de Mantenimiento Centrado en Confiabilidad y la utilización de Python en el tratamiento de los datos obtenidos de SAP, se hace una revisión a las fallas que fueron reportadas durante un semestre (enero - junio 2021) en la planta de Filtrado Larox, con el fin de verificar en que equipos ocurre una mayor cantidad de detenciones y si se tienen medidas preventivas para evitarlas. En este primer análisis, se observó que la mayoría de las fallas ocurrieron tanto en el filtro Larox como en las bombas, ambos equipos críticos para el proceso productivo, por lo que se revisaron los actuales planes de mantenimiento y se informó a la superintendencia de confiabilidad de la división que fallas no tenían un plan de inspección o mantenimiento predictivo asociado a estas detenciones, con el objeto de asignarles alguna actividad que pueda mitigar este riesgo.

Luego de reportar a confiabilidad sobre las fallas que no tenían un plan de mantenimiento activo, se utilizó el Sistema Operativo de Mantenimiento de Codelco y su respectiva matriz de criticidad para clasificar la importancia de cada falla, asignando tres niveles (impacto alto, medio o bajo), lo que se utilizó como base para los datos de entrenamiento del algoritmo de pruebas de Machine Learning. En este algoritmo se probaron cinco técnicas de clasificación, entregando un mayor f1-score (80 %) el método de Bosques Aleatorios, por lo que fue el elegido para el modelo final.

Al definir la técnica a utilizar se programó el algoritmo final, el cual una vez ejecutado presenta dos entregables, el primero un reporte de la cantidad de avisos por ubicación tanto en una tabla como en un gráfico, y el segundo una planilla de cada aviso con su respectivo nivel de criticidad asignado automáticamente. Al revisar manualmente cada predicción de datos nuevos con el real impacto que presenta cada posible falla, se obtiene una precisión de un 81 %, lo cual fue concordante con lo obtenido en la fase de pruebas.

Predecir el impacto de estos avisos presenta un apoyo para el área de planificación y mayor eficiencia en la programación del mantenimiento, por lo que queda como desafío para la empresa la recopilación de más datos y alimentar de más información al algoritmo, para posteriormente evaluar nuevamente las técnicas de Machine Learning y elegir la que presente mayor precisión, perfeccionando el modelo del código final.

Referencias

- [1] CODELCO - CONSEJO MINERO (S.F). CONSEJO MINERO. <https://consejominero.cl/nosotros/socios/codelco/> 3.1.1
- [2] DIVISIÓN ANDINA (S.F). CODELCO. https://www.codelco.com/division-andina/prontus_codelco/2011-02-25/115054.html 3,1,2
- [3] NELSON PIZARRO CONTADOR, PRESIDENTE EJECUTIVO CODELCO CHILE , *Política corporativa del mantenimiento* 3.4.1
- [4] CODELCO (2018). *Sistema Operativo de Mantenimiento*. 3.4.2, 3.4.2.1, 3.4.2.1, 3.4.2.1
- [5] CAMPOS, GONZALO ET AL. (2018) , *Metodología de mantenimiento centrado en confiabilidad (RCM) considerando taxonomía de equipos, base de datos y criticidad de efectos* 4.1
- [6] JOHN MOUBRAY , *Mantenimiento Centrado en Confiabilidad*, Segunda Edición. 4.1, 4.1.1
- [7] JORDI TORRES , *Python Deep Learning* 4.2, 4.2.1.1
- [8] GUSTAVO A. BETANCOURT (2005) , *Las Máquinas de Soporte Vectorial (SVMs)* 4.2.2.1, 4.2.2.1
- [9] SUPPORT VECTOR MACHINE (SVM) - MACHINE LEARNING <https://www.soloentendidos.com/support-vector-machine-machine-learning-2099> 4.2.2.1
- [10] ROCÍO ERANDI BARRIENTOS MARTINEZ ET AL. (2009) , *Árboles de decisión como herramienta en el diagnóstico médico* (document), 4.2.2.2, 4.6, 4.2.2.2
- [11] RANDOM FOREST (BOSQUE ALEATORIO: COMBINANDO ÁRBOLES). IARTIFICIAL. <https://www.iartificial.net/random-forest-bosque-aleatorio/> (document), 4.2.2.3, 4.7
- [12] ARBOLES DE DECISION, RANDOM FOREST, GRADIENT BOOSTING Y C5.0 https://www.cienciadedatos.net/documentos/33_arboles_de_decision_aggregando_random_forest_boosting 1, 2
- [13] NAIVE BAYES - TEORÍA - APRENDE IA <https://aprendeia.com/naive-bayes-teoria-machine-learning/> 4.2.2.4

- [14] ALGORITMO K-NEAREST NEIGHBOR | APRENDE MACHINE LEARNING
<https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/>
 4.2.2.5
- [15] PRECISION, RECALL, F1, ACCURACY EN CLASIFICACIÓN - IARTIFICIAL.NET <https://www.iartificial.net/precision-recall-f1-accuracy-en-clasificacion/> 4.2.3.1, 4.2.3.2, 4.2.3.3, 4.2.3.4
- [16] MIGUEL ÁNGEL CAMACHO ÁLVAREZ & ERNESTO NAVARRO ÁLVAREZ (2020) , *Natural language processing with Python* 4.2.4
- [17] LIBERATORE, GUSTAVO ET AL. (2018) , *Desarrollo de una herramienta para el análisis y representación semántica de colecciones documentales a través del factor TF-IDF* 4.2.4.1
- [18] ¿QUÉ ES PYTHON? <https://web.archive.org/web/20200224120525/https://luca-d3.com/es/data-speaks/diccionario-tecnologico/python-lenguaje> 4.3
- [19] JUPYTER NOTEBOOK: LA HERRAMIENTA DE PYTHON PARA PROCESAR DATOS - IONOS
<https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/jupyter-notebook/> 4.3.1
- [20] LA LIBRERÍA NUMPY <https://aprendeconalf.es/docencia/python/manual/numpy/> 2
- [21] NLTK :: NATURAL LANGUAGE TOOLKIT <https://www.nltk.org> 4
- [22] ¿QUÉ SON LAS STOP WORDS Y PARA QUÉ SIRVEN? - BLACKBEAST <https://blackbeast.pro/diccionario/stop-words/>
 4
- [23] AURÉLIEN GÉRON , *Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow*, Segunda Edición.

A | Anexo I: Categorización avisos inicial

MODO DE FALLA	P1 (Seguridad)	P2 (Medio Ambiente)	P3 (Producción)	P4 (Costos de Mantenimiento)	Puntaje Evaluación	Indicador de Consecuencia	Indicador de Frecuencia	Total	Nivel de Gravedad
ROTURA DE TELA	0	0	2	1	3	1	5	5	2
FALTA DE TENSION DE TELA	0	0	2	1	3	1	3	3	1
SE DEBE CAMBIAR LA JUNTA	0	0	4	1	5	2	3	6	2
PAQUETE DE PLACAS DESALINEADO	0	0	4	1	5	2	4	8	3
PAQUETE DE LUBRICACION	0	0	2	1	3	1	4	4	1
FALTA DE VALVULA	0	0	4	1	5	2	5	10	3
SE PRESENTA FUGA EN VAT PLATE	0	0	4	1	5	2	4	8	3
FUGA DE ACEITE	0	0	2	1	3	1	3	3	1
FISSURA DE BOMBA	0	0	4	1	5	2	3	6	2
REGIRUJACION INTERNA	0	0	4	1	5	2	3	6	2
FALTA DE VALVULA	0	0	8	1	9	2	3	6	2
LA BOMBA SE ENCUENTRA EMBANCADA	0	0	2	1	3	1	5	5	2
SE DEBE CAMBIAR VALVULA	0	0	2	1	3	1	3	3	1
SE DEBE CAMBIAR LA EMPAQUETADURA	0	0	2	1	3	1	5	5	2
ROTURA DE LINEA DE DESCARGA	0	0	4	1	5	2	4	8	3
SE DEBE CAMBIAR LA CAMISA	0	0	4	1	5	2	3	6	2
FALTA DE CORREAS	0	0	2	1	3	1	4	4	1
LAS POLEAS SE ENCUENTRAN DESALINEADAS	0	0	4	1	5	2	3	6	2
SE DEBE CAMBIAR ACEITE	0	10	2	1	13	3	3	9	3
ROTURA DE CONO	0	10	4	1	15	3	3	9	3
LA CORREA ESTA DESCENTRADA	0	0	2	1	3	1	4	4	1
SE DEBE RETIRAR O REPONER LA ESTACION DE POLINES	0	0	4	1	5	2	4	8	3
SE DEBEN REPARAR ESTRUCTURAS, LATERALES Y RASPADORES	0	0	0	1	1	1	3	3	1
FALTA DE RODAMIENTOS	0	0	8	1	9	2	4	8	3

B Anexo II: Resultado de predicción

Ubicación	Descripción	Nivel de importancia
BOMBA N° 4 ESPESADOR CONCENTRADO N°2	Ducto de alimentación con daño	ALTA
SISTEMA INSTRUMENTACIÓN HRO DELKOR	Instalación VV ARMERO DELKOR	ALTA
PLANTA DE FILTRADO LAROX	Cambio de Correa 11D	ALTA
VALVULAS FILTRO LAROX	CAMBIO VV V2-V12	ALTA
SISTEMAS ELÉCTRICOS	Trabaos Habilitación UH LAROX	ALTA
BOMBA N° 4 ESPESADOR CONCENTRADO N°2	ROTURA DE LINEA	ALTA
UNIDAD HIDRAULICA FILTRO LAROX	TENDIDO MANG HIDRAULICAS NUEVA UNIDAD.	ALTA
SISTEMAS ELÉCTRICOS	S#43 SIEMENS Instalar iluminación correa	ALTA
PLANTA DE FILTRADO LAROX	CAMB.ELEM.ELAST.ACOPLE.MOTOR.LAROX	ALTA
LINEAS DE AGUA	Reparo línea alim. agua tl lav. manifold	ALTA
BOMBA N°1 ALIMENTACIÓN LAROX	Bba 1 alimentación larox baja eficiencia	ALTA
HARNERO DELKOR PLANTA LAROX	Línea drenaje harnero delkor embancada	ALTA
SIST BOMBEO BOMBA N°2 LAVADO TELA	bomba 2 con fuga	ALTA
VALVULA V12 FILTRO LAROX	válvula pasandose	ALTA
HARNERO DELKOR PLANTA LAROX	Cambio de rodillo	MEDIA
SISTEMA ALIMBRADO PLANTA LAROX	Inst o Cambiar Focos sobre Bbas Manifold	MEDIA
ESPESADOR DE CONCENTRADO N°2	Cambio De Rastras Espesador #2	MEDIA
BOMBA N°2 LAVADO TELA	Bomba 2 lavado de tela con fuga	MEDIA
BOMBA N°2 LAVADO MANIFOLD	Cambio bombas Grundfos CR-45-4	MEDIA
CORREA DE DESCARGA 72	Cambio de Polines correa 72	MEDIA
SISTEMAS ELÉCTRICOS	INSUMOS TRABAJOS FILTRO LAROX	BAJA
MOTOR ELE SIST TRANSMISION CORREA 72	Limpiaar Motor Correa 72 Mucho Concentrad	BAJA

C | Anexo III: RCM

[illegible]

D | Anexo IV: Código de prueba de técnicas

NLP_PRUEBAS

August 7, 2022

```
[2]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk
from nltk.stem.porter import *
import re
nltk.download('rslp')
nltk.download('stopwords')
pd.options.display.max_columns= None
pd.options.display.max_rows = None
from sklearn import metrics

[nltk_data] Downloading package rslp to
[nltk_data] /Users/sebastianrivasm/nltk_data...
[nltk_data] Package rslp is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data] /Users/sebastianrivasm/nltk_data...
[nltk_data] Package stopwords is already up-to-date!

[3]: #Stemming leaves only the root of the word.
stemmer = PorterStemmer()
#Create set of stopwords
stopwords = nltk.corpus.stopwords.words('spanish')
```

0.1 Limpieza base de datos

```
[4]: data = pd.read_excel("train.xlsx")

[5]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 427 entries, 0 to 426
Data columns (total 13 columns):
#   Column              Non-Null Count  Dtype
---  ---
0   Prioridad           333 non-null    float64
1   Autor del aviso     333 non-null    object
2   Aviso               333 non-null    float64
3   Orden              333 non-null    float64
```

```

4 Ubicación técnica 427 non-null object
5 Descripción      427 non-null object
6 Inicio avería    333 non-null datetime64[ns]
7 Unnamed: 7       0 non-null float64
8 nivel            427 non-null int64
9 Unnamed: 9       202 non-null object
10 Unnamed: 10     0 non-null float64
11 Unnamed: 11     3 non-null object
12 Unnamed: 12     3 non-null float64
dtypes: datetime64[ns](1), float64(6), int64(1), object(5)
memory usage: 43.5+ KB

```

```
[6]: data = data[["Descripción", "nivel"]]
data.head()
```

```
[6]:
```

	Descripción	nivel
0	Cambiar Tela Filtrante	2
1	FABRICACION GREATING CA	3
2	Cambiar Juanta Elastica Motor HM8 Larox	2
3	Rellenar aceite filtro larox	3
4	centrado correa 72	1

```
[7]: data.shape
```

```
[7]: (427, 2)
```

```
[8]: data['nivel'].value_counts()
```

```
[8]: 3    223
      2    110
      1     94
      Name: nivel, dtype: int64
```

```
[9]: X = data['Descripción']
y = data["nivel"]
```

```
[10]: X.shape
```

```
[10]: (427,)
```

```
[11]: y.shape
```

```
[11]: (427,)
```

```
## Preprocesamiento de data
```

```
[12]: X_limpio = []
for i in X:
    for carac in ['.', ',', ':', ';', '!', '?', '|', '@']:
```

```

        i=i.lower().replace(carac,"")
        lineanueva=""
        for pal in i.split():
            if pal not in stopwords:
                lineanueva=lineanueva+stemmer.stem(pal)+" "
        X_limpio.append(lineanueva)
    #print (X_preprocessed)
    X_limpio=np.asarray(X_limpio)

```

```
[13]: #X_limpio
```

0.2 Vectorizacion TF-IDF

```
[14]: from sklearn.feature_extraction.text import CountVectorizer
      from sklearn.feature_extraction.text import TfidfTransformer
      from sklearn.pipeline import Pipeline

      preprocessing = Pipeline([
          ('vect', CountVectorizer()),
          ('tfidf', TfidfTransformer())
      ])

      print("Preprocesando data...")
      X_preprocessed = preprocessing.fit_transform(X_limpio)

```

Preprocesando data...

```
[15]: X_preprocessed.shape
```

```
[15]: (427, 534)
```

0.3 Dividiendo datos de entrenamiento y de prueba

```
[16]: from sklearn.model_selection import train_test_split

[17]: X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y,
    ↪ test_size=0.2, random_state=1)

[18]: X_train.shape

[18]: (341, 534)

[19]: y_test.shape

[19]: (86,)

[20]: #DTy_test = pd.DataFrame(y_test)
      #DTy_test.reset_index(drop = True, inplace = True)

```

```
[21]: #DTy_test.head()

[22]: #DTx_test = pd.DataFrame(X_test)
      #DTx_test.reset_index(drop = True, inplace = True)

[23]: #DTx_test.head()

[24]: #DTtest = DTx_test.join(DTy_test)

[25]: #DTtest
```

1 Probando con Maquina Soporte Vectorial (SVM)

```
[26]: from sklearn.svm import SVC
      from sklearn.metrics import f1_score

[27]: modelSVC = SVC()

[28]: modelSVC.fit(X_train,y_train)

[28]: SVC()

[28]: pred = modelSVC.predict(X_test)

[33]: from sklearn.metrics import classification_report, confusion_matrix

[30]: print(confusion_matrix(y_test, pred))
      print(classification_report(y_test,pred))

[[ 8  0 10]
 [ 0  9 10]
 [ 0  0 49]]
              precision    recall  f1-score   support

         1         1.00      0.44      0.62         18
         2         1.00      0.47      0.64         19
         3         0.71      1.00      0.83         49

 accuracy          0.86
 macro avg          0.90
weighted avg          0.83
```

```
[31]: f1_score(y_test, pred, average='weighted')

[31]: 0.7440246982028613
```

```
[32]: print(metrics.accuracy_score(y_test, pred))

0.7674418604651163

[33]: from sklearn.model_selection import GridSearchCV

[34]: param_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001]}

[35]: grid = GridSearchCV(SVC(), param_grid, scoring="f1_weighted", verbose = 2)

[36]: grid.fit(X_train, y_train)
```

Fitting 5 folds for each of 25 candidates, totalling 125 fits

```
[CV] END ...C=0.1, gamma=1; total time= 0.0s
[CV] END ...C=0.1, gamma=1; total time= 0.0s
[CV] END ...C=0.1, gamma=1; total time= 0.0s
[CV] END ...C=0.1, gamma=1; total time= 0.0s
[CV] END ...C=0.1, gamma=1; total time= 0.0s
[CV] END ...C=0.1, gamma=0.1; total time= 0.0s
[CV] END ...C=0.1, gamma=0.1; total time= 0.0s
[CV] END ...C=0.1, gamma=0.1; total time= 0.0s
[CV] END ...C=0.1, gamma=0.1; total time= 0.0s
[CV] END ...C=0.1, gamma=0.1; total time= 0.0s
[CV] END ...C=0.1, gamma=0.01; total time= 0.0s
[CV] END ...C=0.1, gamma=0.01; total time= 0.0s
[CV] END ...C=0.1, gamma=0.01; total time= 0.0s
[CV] END ...C=0.1, gamma=0.01; total time= 0.0s
[CV] END ...C=0.1, gamma=0.01; total time= 0.0s
[CV] END ...C=0.1, gamma=0.001; total time= 0.0s
[CV] END ...C=0.1, gamma=0.001; total time= 0.0s
[CV] END ...C=0.1, gamma=0.001; total time= 0.0s
[CV] END ...C=0.1, gamma=0.001; total time= 0.0s
[CV] END ...C=0.1, gamma=0.001; total time= 0.0s
[CV] END ...C=0.1, gamma=0.0001; total time= 0.0s
[CV] END ...C=0.1, gamma=0.0001; total time= 0.0s
[CV] END ...C=0.1, gamma=0.0001; total time= 0.0s
[CV] END ...C=0.1, gamma=0.0001; total time= 0.0s
[CV] END ...C=0.1, gamma=0.0001; total time= 0.0s
[CV] END ...C=1, gamma=1; total time= 0.0s
[CV] END ...C=1, gamma=1; total time= 0.0s
[CV] END ...C=1, gamma=1; total time= 0.0s
[CV] END ...C=1, gamma=1; total time= 0.0s
[CV] END ...C=1, gamma=1; total time= 0.0s
[CV] END ...C=1, gamma=0.1; total time= 0.0s
[CV] END ...C=1, gamma=0.1; total time= 0.0s
[CV] END ...C=1, gamma=0.1; total time= 0.0s
[CV] END ...C=1, gamma=0.1; total time= 0.0s
[CV] END ...C=1, gamma=0.1; total time= 0.0s
```

6

```
[36]: GridSearchCV(estimator=SVC(),
                    param_grid={'C': [0.1, 1, 10, 100, 1000],
                                'gamma': [1, 0.1, 0.01, 0.001, 0.0001]},
                    scoring='f1_weighted', verbose=2)
```

```
[37]: grid.best_params_
[37]: {'C': 10, 'gamma': 0.1}
[38]: grid_pred = grid.predict(X_test)
[39]: print(confusion_matrix(y_test, grid_pred))
print(classification_report(y_test, grid_pred))
```

```
[[10  0  8]
 [ 1 11  7]
 [ 4  1 44]]
```

	precision	recall	f1-score	support
1	0.67	0.56	0.61	18
2	0.92	0.58	0.71	19
3	0.75	0.90	0.81	49
accuracy			0.76	86
macro avg	0.78	0.68	0.71	86
weighted avg	0.77	0.76	0.75	86

```
[40]: f1_score(y_test, grid_pred, average='weighted')
[40]: 0.7478940442181252
[41]: print(metrics.accuracy_score(y_test, grid_pred))
0.7558139534883721
[42]: #DTgrid = pd.DataFrame(grid_pred)
#DTgrid.reset_index(drop = True, inplace = True)
#df_gridunido = DTtest.join(DTgrid)
[43]: #df_gridunido
```

2 Probando con Decision Tree

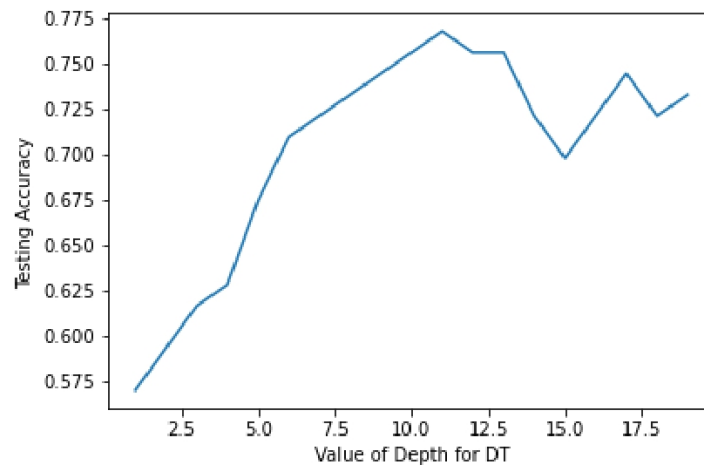
```
[29]: from sklearn.tree import DecisionTreeClassifier
[30]: resultados=[]
depth_range=range(1,20)
for i in depth_range:
    dtree=DecisionTreeClassifier(max_depth=i)
    dtree.fit(X_train, y_train)
    dectre_pred = dtree.predict(X_test)
    resultados.append(metrics.accuracy_score(y_test, dectre_pred))
```

```
# import Matplotlib (scientific plotting library)
import matplotlib.pyplot as plt

# allow plots to appear within the notebook
%matplotlib inline

# plot the relationship between K and testing accuracy
plt.plot(depth_range, resultados)
plt.xlabel('Value of Depth for DT')
plt.ylabel('Testing Accuracy')
```

```
[30]: Text(0, 0.5, 'Testing Accuracy')
```



```
[40]: dtree=DecisionTreeClassifier(max_depth=11)
dtree.fit(X_train,y_train)
dtree_pred = dtree.predict(X_test)
```

```
[41]: print(confusion_matrix(y_test, dtree_pred))
print(classification_report(y_test, dtree_pred))
```

```
[[ 6  0 12]
 [ 0 13  6]
 [ 0  2 47]]
```

```
precision    recall  f1-score   support
```

1	1.00	0.33	0.50	18
2	0.87	0.68	0.76	19
3	0.72	0.96	0.82	49
accuracy			0.77	86
macro avg	0.86	0.66	0.70	86
weighted avg	0.81	0.77	0.74	86

```
[42]: f1_score(y_test, dtree_pred, average='weighted')
[42]: 0.743406052751578
[39]: print(metrics.accuracy_score(y_test, dtree_pred))
0.7441860465116279
## Probando con Random Forest
[50]: from sklearn.ensemble import RandomForestClassifier
[51]: rfcl=RandomForestClassifier(random_state=33)
[52]: param_grid = {
      'n_estimators': [60,200],
      'criterion' :['gini', 'entropy']
    }
[53]: CV_rfc = GridSearchCV(estimator=rfcl, param_grid=param_grid,
      scoring="f1_weighted", cv= 5)
      CV_rfc.fit(X_train, y_train)
[53]: GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=33),
      param_grid={'criterion': ['gini', 'entropy'],
      'n_estimators': [60, 200]},
      scoring='f1_weighted')
[54]: CV_rfc.best_params_
[54]: {'criterion': 'gini', 'n_estimators': 200}
[55]: rfcl=RandomForestClassifier(random_state=33, n_estimators= 200,
      criterion='gini')
[56]: rfcl.fit(X_train, y_train)
[56]: RandomForestClassifier(n_estimators=200, random_state=33)
[57]: predrfc=rfcl.predict(X_test)
```

```
[58]: f1_score(y_test, predrfc, average='weighted')

[58]: 0.7653065289078275

[59]: rfc = RandomForestClassifier(n_estimators = 60, criterion = 'gini',
    ↪ random_state = 33)

[60]: rfc.fit(X_train,y_train)

[60]: RandomForestClassifier(n_estimators=60, random_state=33)

[61]: rfc_pred = rfc.predict(X_test)

[62]: print(confusion_matrix(y_test,rfc_pred))
print(classification_report(y_test,rfc_pred))

[[10  0  8]
 [ 0 13  6]
 [ 1  1 47]]
           precision    recall  f1-score   support

    1         0.91      0.56      0.69         18
    2         0.93      0.68      0.79         19
    3         0.77      0.96      0.85         49

 accuracy          0.81         86
 macro avg          0.87         86
 weighted avg          0.83         86

[63]: f1_score(y_test, rfc_pred, average='weighted')

[63]: 0.8053048528589827

[64]: print(metrics.accuracy_score(y_test, rfc_pred))

0.813953488372093

[65]: #DTrfc = pd.DataFrame(rfc_pred)
#DTrfc.reset_index(drop = True, inplace = True)
#df_rfcunido = DTtest.join(DTrfc)

[66]: #df_rfcunido
```

3 Probando con Naive Bayes

```
[67]: from sklearn.naive_bayes import GaussianNB
```

```
[68]: nb = GaussianNB()
[69]: nb.fit(X_train.toarray(), y_train)
[69]: GaussianNB()
[70]: nb_pred = nb.predict(X_test.toarray())
[71]: print(confusion_matrix(y_test,nb_pred))
print(classification_report(y_test,nb_pred))
```

```
[[11  3  4]
 [ 2 11  6]
 [ 7 14 28]]
```

	precision	recall	f1-score	support
1	0.55	0.61	0.58	18
2	0.39	0.58	0.47	19
3	0.74	0.57	0.64	49
accuracy			0.58	86
macro avg	0.56	0.59	0.56	86
weighted avg	0.62	0.58	0.59	86

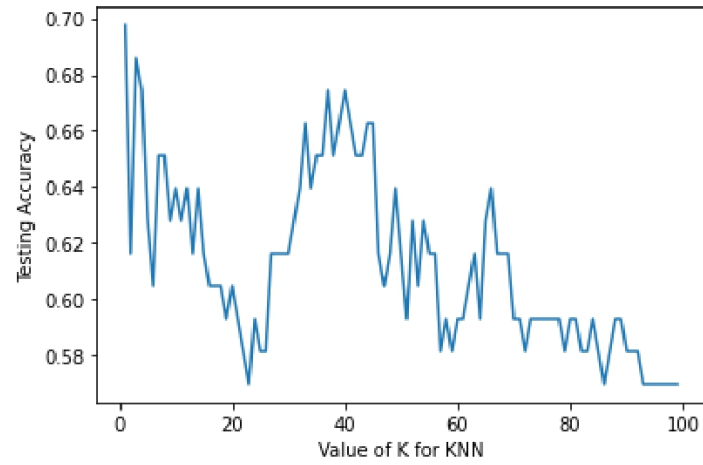
3.1 Probando con KNN

```
[72]: from sklearn.neighbors import KNeighborsClassifier
[73]: k_range = range(1, 100)
scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    pred_knn = knn.predict(X_test)
    scores.append(metrics.accuracy_score(y_test, pred_knn))
[74]: # importar Matplotlib (librería de gráficos)
import matplotlib.pyplot as plt

# Permite que los gráficos sean visibles en el notebook
%matplotlib inline

# Grafica la relación entre K vecinos y Accuracy
plt.plot(k_range, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
```

[74]: Text(0, 0.5, 'Testing Accuracy')



E | Anexo V: Código final

NLP_FINAL

August 7, 2022

```
[ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk
from nltk.stem.porter import PorterStemmer
import re
import six
nltk.download('rslp')
nltk.download('stopwords')
pd.options.display.max_columns = None
pd.options.display.max_rows = None
from sklearn import metrics

[2]: #Stemming leaves only the root of the word.
stemmer = PorterStemmer()
#Create set of stopwords
stopwords = nltk.corpus.stopwords.words('spanish')
```

1 Carga de base de datos inicial

Se cargan las dos bases de datos a utilizar, train que es la base de datos de entrenamiento del algoritmo y SIEMPRE debe estar, y data que es la base de datos que se va actualizando dependiendo del período. OBSERVACIÓN: LA BASE DE DATOS A EVALUAR, DEBE TENER POR NOMBRE “AVISOS” Y DEBE ESTAR EN FORMATO .xlsx, NO .xls.

```
[3]: data_train = pd.read_excel("train.xlsx")

[4]: data = pd.read_excel("AVISOS.xlsx")

[5]: eliminar = ["MTTO", 'Mtto', 'mtto', 'Mant', 'MANT', 'mant', 'prep', 'PREP',
↳ 'Prep']
data = data[~data['Descripción'].str.contains('|'.join(eliminar))]

[6]: data = data.reset_index(level=None, drop=False, inplace=False, col_level=0,
↳ col_fill='')

[7]: data
```

[7]:	index	Prioridad	Inicio de avería	Denominación de la ubicación técnica \
0	0	3	2021-10-27	ESPESADOR DE CONCENTRADO N°2
1	1	3	2021-10-27	SISTEMAS ELÉCTRICOS
2	3	3	2021-10-25	SISTEMA INSTRUMENTACIÓN HRO DELKOR
3	4	3	2021-10-21	PLANTA DE FILTRADO LAROX
4	5	3	2021-10-21	VÁLVULAS FILTRO LAROX
5	6	3	2021-10-21	SISTEMAS ELÉCTRICOS
6	9	3	2021-10-20	BOMBA N° 4 ESPESADOR CONCENTRADO N°2
7	11	3	2021-10-19	UNIDAD HIDRÁULICA FILTRO LAROX
8	12	3	2021-10-15	SISTEMAS ELÉCTRICOS
9	13	3	2021-10-01	BOMBA N°2 LAVADO MANIFOLD
10	14	3	2021-10-17	PLANTA DE FILTRADO LAROX
11	15	3	NaT	BOMBA N° 4 ESPESADOR CONCENTRADO N°2
12	16	3	NaT	BOMBA N°1 ALIMENTACIÓN LAROX
13	18	3	NaT	BOMBA N°2 LAVADO TELA
14	19	3	NaT	SIST BOMBEO BBA N°2 LAVADO TELA
15	20	3	NaT	VÁLVULA V12
16	21	3	NaT	HARNERO DELKOR PLANTA LAROX
17	22	3	NaT	HARNERO DELKOR PLANTA LAROX
18	23	3	NaT	LINEAS DE AGUA
19	24	3	NaT	SISTEMA ALUMBRADO PLANTA LAROX
20	25	3	NaT	MOTOR ELE SIST TRANSMISION CORREA 72
21	26	3	NaT	CORREA 72

	Descripción	Creado el	Aviso \
0	Cambio De Rastras Espesador #2	2021-10-27	24470253
1	INSUMOS TRABAJOS FILTRO LAROX	2021-10-27	24469232
2	Instalacion V/V ARNERO DELKOR	2021-10-25	24461266
3	Cambio de Correa 11D	2021-10-21	24453371
4	CAMBIO V/V V2-V12	2021-10-21	24453283
5	Trabaos Habilitacion U/H LAROX	2021-10-21	24452794
6	ROTURA DE LINEA	2021-10-20	24447361
7	TENDIDO MANG.HIDRAULICAS NUEVA UNIDAD.	2021-10-19	24445459
8	S#43 SIEMENS Instalar iluminación correa	2021-10-13	24427687
9	Cambio bombas Grundfos CR-45-4	2021-10-01	24403474
10	CAMB.ELEM.ELÁST.ACOPLA.MOTOR.LÁROX	2021-10-17	24436160
11	DUcto de alimentacion con daño	2021-10-20	24447364
12	Bba 1 alimentación larox baja eficiencia	2021-10-16	24435223
13	Bomba 2 lavado de tela con fuga	2021-10-16	24435225
14	bomba 2 con fuga	2021-10-04	24408347
15	válvula pasandose	2021-10-23	24456584
16	Cambio de rodillo	2021-10-18	24441589
17	Línea drenaje harnero delkor embancada	2021-10-16	24435224
18	Repara linea alim. agua tk lav. manifold	2021-10-24	24457632
19	Inst o Cambiar Focos sobre Bbas Manifold	2021-10-21	24451561
20	Limpiar Motor Correa 72 Mucho Concentrad	2021-10-21	24451564
21	Cambio de Polines correa 72	2021-10-07	24417048

Orden Creado por Status de usuario Status del sistema \				
0	105152390.0	HERMEIE	APRO	METR ORAS
1	105151308.0	GDON0005	APRO	METR ORAS
2	105143418.0	GDON0005	APRO	METR ORAS
3	105136063.0	HERMEIE	APRO	METR ORAS
4	105136020.0	GDON0005	APRO	METR ORAS
5	105135510.0	GDON0005	APRO	METR ORAS
6	105131902.0	DMARA001	APRO	METR ORAS
7	105128539.0	VVALENZU	APRO	METR ORAS
8	105117476.0	SAVEN003	APRO	METR ORAS
9	105089185.0	HERMEIE	APRO	METR ORAS
10	NaN	GGONZ065	APRO DOCU	MEAB
11	NaN	DMARA001	APRO	METR
12	NaN	CCONT069	APRO	MEAB
13	NaN	CCONT069	APRO	MEAB
14	NaN	LRODR027	APRO	MEAB
15	NaN	LRODR027	APRO	MEAB
16	NaN	DMARA001	APRO	METR
17	NaN	CCONT069	APRO	MEAB
18	NaN	JHERR052	APRO	MEAB
19	NaN	JBARR012	APRO	MEAB
20	NaN	JBARR012	APRO	MEAB
21	NaN	LSANH002	APRO	METR

Hora de inicio de avería Inicio deseado Ubicación técnica			
0	15:33:02	2021-11-03	AOAS-LAR-EDC-E01
1	11:42:04	2021-11-03	AOAS-LAR-SEL
2	12:02:55	2021-11-01	AOAS-LAR-HAR-INS
3	15:37:26	2021-10-28	AOAS-LAR
4	15:28:50	2021-10-28	AOAS-LAR-FLA-VAL
5	12:32:41	2021-10-28	AOAS-LAR-SEL
6	12:22:19	2021-10-27	AOAS-LAR-BPL-B03
7	12:42:18	2021-10-26	AOAS-LAR-FLA-HID
8	11:55:23	2021-10-20	AOAS-LAR-SEL
9	09:25:52	2021-10-08	AOAS-LAR-BPL-B07
10	14:40:00	2021-10-17	AOAS-LAR
11	00:00:00	2021-10-27	AOAS-LAR-BPL-B03
12	00:00:00	2021-10-18	AOAS-LAR-BPL-B04
13	00:00:00	2021-10-18	AOAS-LAR-BPL-B12
14	00:00:00	2021-10-04	AOAS-LAR-BPL-B12-SB0
15	00:00:00	2021-10-23	AOAS-LAR-FLA-VAL-PIN-V05
16	00:00:00	2021-10-25	AOAS-LAR-HAR
17	00:00:00	2021-10-18	AOAS-LAR-HAR
18	00:00:00	2021-10-31	AOAS-LAR-LFL-LO3
19	00:00:00	2021-11-28	AOAS-LAR-SEL-SAE
20	00:00:00	2021-11-28	AOAS-LAR-CTR-C01-STR-M01

21

00:00:00

2021-11-04

AOAS-LAR-CTR-C01

2 Reporte por ubicación

Aquí paso a nombrar cada ubicación técnica por su denominación, ej: 'AOAS-LAR-FLA' pasa a ser 'FILTRO LAROX'

```
[8]: ut = pd.read_excel("ubtec.xlsx")
     ut = ut[["Ubicación técnica", "Denominación de la ubicación técnica"]]
     dicut = dict(ut.values.tolist())

[9]: def map_values(row, values_dict):
     return values_dict[row]
     data["Ubicación"] = data["Ubicación técnica"].apply(map_values, args=(dicut,))

[10]: data_cuenta = data["Ubicación"].value_counts().rename_axis('Ubicación').
      ↪reset_index(name='Cantidad de avisos')

[11]: def render_mpl_table(data, col_width=3.0, row_height=0.625, font_size=14,
      ↪header_color='#40466e', row_colors=['#f1f1f2', 'w'],
      ↪Edge_color='w',
      ↪bbox=[0, 0, 1, 1], header_columns=0,
      ↪ax=None, **kwargs):
     if ax is None:
         size = (np.array(data.shape[:-1]) + np.array([0, 1])) * np.
      ↪array([col_width, row_height])
         fig, ax = plt.subplots(figsize=size)
         ax.axis('off')

         mpl_table = ax.table(cellText=data.values, bbox=bbox, colLabels=data.
      ↪columns, **kwargs)

         mpl_table.auto_set_font_size(False)
         mpl_table.set_fontsize(font_size)

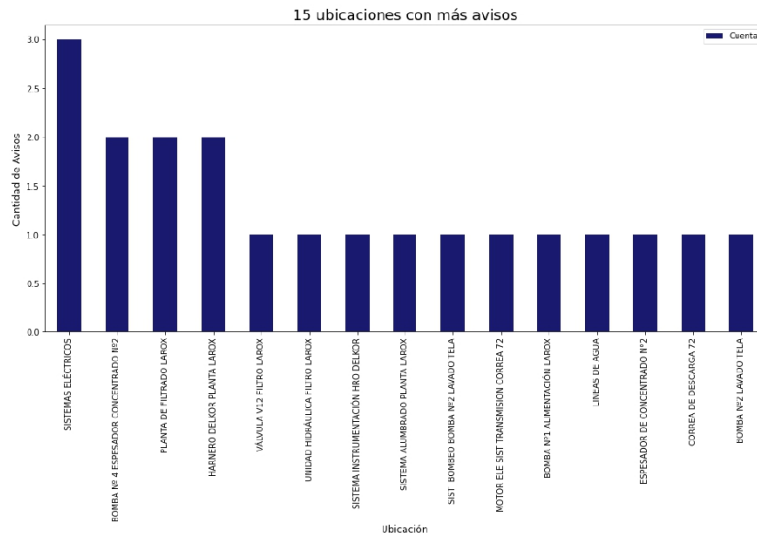
         for k, cell in six.iteritems(mpl_table._cells):
             cell.set_edgecolor(Edge_color)
             if k[0] == 0 or k[1] < header_columns:
                 cell.set_text_props(weight='bold', color='w')
                 cell.set_facecolor(header_color)
             else:
                 cell.set_facecolor(row_colors[k[0]%len(row_colors) ])
         return fig.savefig('Cantidad de avisos por ubicación.jpg')

render_mpl_table(data_cuenta, header_columns=0, col_width=7.0)
```

Ubicación	Cantidad de avisos
SISTEMAS ELÉCTRICOS	3
PLANTA DE FILTRADO LAROX	2
BOMBA N° 4 ESPESADOR CONCENTRADO N°2	2
HARNERO DELKOR PLANTA LAROX	2
ESPESADOR DE CONCENTRADO N°2	1
SIST BOMBEO BOMBA N°2 LAVADO TELA	1
MOTOR ELE SIST TRANSMISION CORREA 72	1
SISTEMA ALUMBRADO PLANTA LAROX	1
LINEAS DE AGUA	1
VÁLVULA V12 FILTRO LAROX	1
BOMBA N°1 ALIMENTACIÓN LAROX	1
BOMBA N°2 LAVADO TELA	1
BOMBA N°2 LAVADO MANIFOLD	1
UNIDAD HIDRÁULICA FILTRO LAROX	1
VÁLVULAS FILTRO LAROX	1
SISTEMA INSTRUMENTACIÓN HRO DELKOR	1
CORREA DE DESCARGA 72	1

```
[12]: data["Cuenta"] = 1

[13]: grafico = pd.pivot_table(data,index="Ubicación", values="Cuenta",aggfunc='sum')
grafico.sort_values("Cuenta", ascending=False, inplace=True)
grafico.iloc[0:15].plot(kind='bar', color = 'midnightblue', figsize=(16,7))
plt.title("15 ubicaciones con más avisos", fontsize=18)
plt.xlabel("Ubicación", fontsize=12)
plt.ylabel("Cantidad de Avisos", fontsize=12)
plt.savefig("Ubicaciones con más avisos.jpg",bbox_inches = 'tight') #guardar_
↪ imagen
```



3 Entrenamiento con Random Forest

```
[14]: data_train = data_train[["Descripción", "nivel"]]

[15]: X = data_train['Descripción']
      y = data_train["nivel"]
      X_pred = data["Descripción"]

[16]: X_limpio = []
      for i in X:
          for carac in ['.', ',', ':', ';', '!', '?', '|', '@']:
              i = i.lower().replace(carac, "")
              lineanueva = ""
              for pal in i.split():
                  if pal not in stopwords:
                      lineanueva = lineanueva + stemmer.stem(pal) + " "
              X_limpio.append(lineanueva)
      #print (X_preprocessed)
      X_limpio = np.asarray(X_limpio)

      X_predvec = []
```

```

for i in X_pred:
    for carac in ['.', ',', ':', ';', '!', '?', '|', '@']:
        i=i.lower().replace(carac,"")
        lineanueva=""
    for pal in i.split():
        if pal not in stopwords:
            lineanueva=lineanueva+stemmer.stem(pal)+" "
    X_predvec.append(lineanueva)
#print (X_preprocessed)
X_predvec=np.asarray(X_predvec)

```

```
[ ]:
```

```

[17]: from sklearn.feature_extraction.text import CountVectorizer
      from sklearn.feature_extraction.text import TfidfTransformer
      from sklearn.pipeline import Pipeline

      preprocessing = Pipeline([
          ('vect', CountVectorizer()),
          ('tfidf', TfidfTransformer())
      ])

      print("Preprocesando data...")
      X_preprocessed = preprocessing.fit_transform(X_limpio)

      print("Preprocessing test data...")
      Xpred_preprocessed = preprocessing.transform(X_predvec)

```

```

Preprocesando data...
Preprocessing test data...

```

```
[18]: X_preprocessed.shape
```

```
[18]: (427, 534)
```

```
[19]: Xpred_preprocessed.shape
```

```
[19]: (22, 534)
```

```
[20]: from sklearn.ensemble import RandomForestClassifier
```

```

[21]: rfc = RandomForestClassifier(n_estimators = 200, criterion = 'gini',
      ↪ random_state = 33)

```

```
[22]: rfc.fit(X_preprocessed,y)
```

```
[22]: RandomForestClassifier(n_estimators=60, random_state=33)
```

```

[23]: rfc_pred = rfc.predict(Xpred_preprocessed)

[24]: data = data.loc[:, {'Ubicación', 'Descripción'}]
      data = data[['Ubicación', 'Descripción']]

[25]: DFpred = pd.DataFrame(rfc_pred)
      DFpred.reset_index(drop = True, inplace = True)
      df_pred = data.join(DFpred)

[26]: df_pred = df_pred.rename(columns={0: 'Nivel de importancia'})

[27]: df_pred = df_pred.sort_values('Nivel de importancia', ascending=False)

[28]: df_pred = df_pred.replace({3: "ALTA", 2: "MEDIA", 1: "BAJA"})

[29]: df_pred.to_excel('Predicción de Avisos.xlsx', index = False)

[30]: df_pred.head()

```

	Ubicación	Descripción \
11	BOMBA N° 4 ESPESADOR CONCENTRADO N°2	DUcto de alimentacion con daño
2	SISTEMA INSTRUMENTACIÓN HRO DELKOR	Instalacion V/V ARNERO DELKOR
3	PLANTA DE FILTRADO LAROX	Cambio de Correa 11D
4	VÁLVULAS FILTRO LAROX	CAMBIO V/V V2-V12
5	SISTEMAS ELÉCTRICOS	Trabaos Habilitacion U/H LAROX

	Nivel de importancia
11	ALTA
2	ALTA
3	ALTA
4	ALTA
5	ALTA

```

[31]: def render2_mpl_table(data, col_width=3.0, row_height=0.625, font_size=14,
      header_color='#40466e', row_colors=['#f1f1f2', 'w'],
      ↪Edge_color='w',
      bbox=[0, 0, 1, 1], header_columns=0,
      ax=None, **kwargs):
    if ax is None:
        size = (np.array(data.shape[:-1]) + np.array([0, 1])) * np.
        ↪array([col_width, row_height])
        fig, ax = plt.subplots(figsize=size)
        ax.axis('off')

        mpl_table = ax.table(cellText=data.values, bbox=bbox, colLabels=data.
        ↪columns, **kwargs)

```

```

mpl_table.auto_set_font_size(False)
mpl_table.set_fontsize(font_size)

for k, cell in six.iteritems(mpl_table._cells):
    cell.set_edgecolor(Edge_color)
    if k[0] == 0 or k[1] < header_columns:
        cell.set_text_props(weight='bold', color='w')
        cell.set_facecolor(header_color)
    else:
        cell.set_facecolor(row_colors[k[0]%len(row_colors) ])
return fig.savefig('Prediccion.jpg')

render2_mpl_table(df_pred, header_columns=0, col_width=7.0)

```

Ubicación	Descripción	Nivel de importancia
BOMBA N° 4 ESPESADOR CONCENTRADO N°2	Ducto de alimentación con daño	ALTA
SISTEMA INSTRUMENTACIÓN HRO DELKOR	Instalación V/V ARNERO DELKOR	ALTA
PLANTA DE FILTRADO LAROX	Cambio de Correa 11D	ALTA
VÁLVULAS FILTRO LAROX	CAMBIO V/V V2-V12	ALTA
SISTEMAS ELÉCTRICOS	Trabajos Habilitación U/H LAROX	ALTA
BOMBA N° 4 ESPESADOR CONCENTRADO N°2	ROTURA DE LÍNEA	ALTA
UNIDAD HIDRÁULICA FILTRO LAROX	TENDIDO MANG.HIDRAULICAS NUEVA UNIDAD.	ALTA
SISTEMAS ELÉCTRICOS	S#43 SIEMENS Instalar iluminación correa	ALTA
PLANTA DE FILTRADO LAROX	CAMB.ELEM.ELÁST.ACOPLE.MOTOR.LAROX	ALTA
LÍNEAS DE AGUA	Repara línea alim. agua tk lav. manifold	ALTA
BOMBA N°1 ALIMENTACIÓN LAROX	Bba 1 alimentación larox baja eficiencia	ALTA
HARNERO DELKOR PLANTA LAROX	Línea drenaje harnero delkor embancada	ALTA
SIST BOMBEO BOMBA N°2 LAVADO TELA	bomba 2 con fuga	ALTA
VÁLVULA V12 FILTRO LAROX	válvula pasandose	ALTA
HARNERO DELKOR PLANTA LAROX	Cambio de rodillo	MEDIA
SISTEMA ALUMBRADO PLANTA LAROX	Inst o Cambiar Focos sobre Bbas Manifold	MEDIA
ESPESADOR DE CONCENTRADO N°2	Cambio De Rastras Espesador #2	MEDIA
BOMBA N°2 LAVADO TELA	Bomba 2 lavado de tela con fuga	MEDIA
BOMBA N°2 LAVADO MANIFOLD	Cambio bombas Grundfos CR-45-4	MEDIA
CORREA DE DESCARGA 72	Cambio de Polines correa 72	MEDIA
SISTEMAS ELÉCTRICOS	INSUMOS TRABAJOS FILTRO LAROX	BAJA
MOTOR ELE SIST TRANSMISIÓN CORREA 72	Limpiar Motor Correa 72 Mucho Concentrad	BAJA

[]: