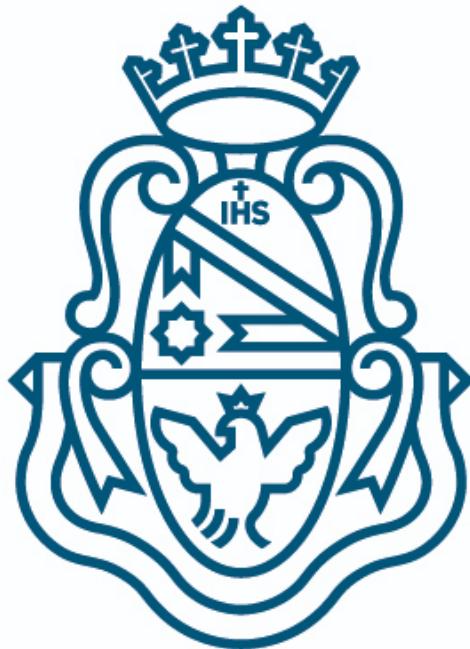


UNIVERSIDAD NACIONAL DE CÓRDOBA  
Facultad de Ciencias Exactas, Físicas y Naturales



PROYECTO FINAL INTEGRADOR

**“Predicción de cantidad de defectos graves en  
vehículos utilitarios en planta automotriz”**

[Enlace al repositorio](#)

Gerardo A. COLLANTE  
Matrícula: 39.022.782  
Email: gerardo.collante@unc.edu.ar  
Cel: 54 (03574) 650490

**Supervisor**  
Dr. Ing. Orlando MICOLINI  
24 de noviembre de 2021

## Agradecimientos

*En primer lugar quiero agradecer profundamente a todas las personas que formaron parte de este viaje que culmina. Cada una de ellas aportó su granito de arena para que hoy me sea posible hacer realidad el objetivo más importante de mi vida. Especialmente a mis papás Daniel y Gloria, que ofreciendo su tiempo y esfuerzo me concedieron la dicha de poder formarme como profesional.*

*También quiero agradecer a mi supervisor Dr. Ing. Orlando Micolini por impartir conocimiento, brindar consejos y transmitir su experiencia a lo largo de todo el proceso del proyecto integrador.*

*Finalmente, el reconocimiento a la Universidad Nacional de Córdoba y la Facultad de Ciencias Exactas, Físicas y Naturales por estos invaluos años de saber y formación. La universidad pública me proveyó los recursos para que en el presente día me encuentre en condiciones de finalizar mi carrera de grado y graduarme como ingeniero, en virtud de ello mi gratitud eterna.*

# Índice general

<b>1 Introducción</b>	<b>5</b>
1.1 Motivación . . . . .	5
1.2 Descripción general . . . . .	5
1.3 Objetivos . . . . .	7
1.3.1 Objetivos principales . . . . .	7
1.3.2 Objetivos secundarios . . . . .	7
1.4 Requerimientos . . . . .	8
1.4.1 Requerimientos funcionales . . . . .	8
1.4.2 Requerimientos no funcionales . . . . .	9
1.5 Riesgos . . . . .	10
1.5.1 Tipos de riesgos . . . . .	10
1.6 Gestión de riesgos . . . . .	10
1.6.1 Identificación de los riesgos . . . . .	10
1.6.2 Análisis de riesgos . . . . .	12
1.6.3 Selección de riesgos . . . . .	13
1.6.4 Elaboración de planes de riesgos . . . . .	13
<b>2 Inteligencia Artificial</b>	<b>15</b>
2.1 Aprendizaje automático . . . . .	15
<b>3 Procesamiento de datos</b>	<b>16</b>
3.1 Origen de los datos . . . . .	16
3.1.1 Organización de la usina . . . . .	16
3.1.2 Nomenclatura de defectos . . . . .	17
3.1.3 Gravedad . . . . .	17
3.1.4 Tipo . . . . .	17
3.1.5 Puntos de reconocimiento de defectos . . . . .	17
3.2 Ingeniería de datos . . . . .	18
3.2.1 Recolección de datos . . . . .	19
3.2.2 Limpieza de datos . . . . .	19
3.2.3 Preprocesamiento de datos . . . . .	19
3.2.4 Análisis y visualización de datos . . . . .	19
3.2.5 Imputación de datos . . . . .	26
3.2.6 Formateo de datos . . . . .	36
3.2.7 Pipeline de datos . . . . .	44

<b>4 Modelos</b>	<b>48</b>
4.1 Métricas . . . . .	48
4.1.1 MAPE . . . . .	48
4.1.2 MASE . . . . .	48
4.2 <i>Framework</i> . . . . .	50
4.2.1 Callbacks . . . . .	51
4.3 Arquitecturas . . . . .	51
4.3.1 Hiperpárametros de capas . . . . .	52
4.3.2 Tipos . . . . .	53
4.4 <b>Entrenamiento</b> . . . . .	55
4.4.1 Datos de prueba . . . . .	55
4.4.2 Desplazamiento temporal . . . . .	57
4.4.3 Tipo de horizonte . . . . .	58
4.4.4 Resultados . . . . .	60
4.5 Ajuste de hiperparámetros . . . . .	64
4.5.1 <i>Keras-tuner</i> . . . . .	65
4.5.2 Espacio de búsqueda . . . . .	66
4.5.3 Resultados <i>dataset</i> polución . . . . .	67
4.5.4 Conclusión . . . . .	71
4.5.5 Resultados <i>dataset</i> FSI . . . . .	71
4.5.6 Cambio en preprocesamiento de datos . . . . .	76
4.5.7 Conclusión . . . . .	79
4.6 <b>Entropía</b> . . . . .	79
4.6.1 Rendimiento de los algoritmos . . . . .	80
4.6.2 <i>Dataset</i> de polución . . . . .	81
4.6.3 <i>Dataset</i> FSI . . . . .	81
4.6.4 Comparación de <i>datasets</i> . . . . .	82
<b>5 Despliegue</b>	<b>84</b>
5.1 Áreas fundamentales a tener en cuenta para un proyecto de inteligencia artificial . . . . .	84
5.1.1 Almacenamiento y extracción de datos . . . . .	84
5.1.2 <i>Frameworks</i> y herramientas . . . . .	85
5.1.3 <i>Feedback</i> e iteración . . . . .	86
5.2 Análisis del problema . . . . .	86
5.2.1 Almacenamiento y extracción de datos . . . . .	86
5.2.2 <i>Frameworks</i> y herramientas . . . . .	87
5.2.3 <i>Feedback</i> e iteración . . . . .	87
5.3 Arquitectura propuesta . . . . .	87
5.3.1 <i>Frameworks</i> , herramientas y <i>software</i> . . . . .	87
5.3.2 Descripción . . . . .	88

<b>6 Conclusiones</b>	<b>90</b>
-----------------------	-----------

# 1 Introducción

## 1.1 Motivación

Las motivaciones para el desarrollo de este proyecto pueden separarse en dos. Por un lado aquellas relacionadas al producto en sí, destacamos la posibilidad de brindar una solución en un entorno industrial y por consiguiente de alto impacto tanto a nivel técnico como económico. Además será de gran valor a nuestro perfil profesional el hecho de crear software de calidad dentro de un ambiente laboral.

Por el otro lado, una vasta cantidad de campos científicos están obteniendo avances sustanciales debido al uso de algoritmos de inteligencia artificial. Más allá del marketing que envuelve a la materia, es indudable que su implementación está impactando de forma directa en diversos sectores sin limitarse exclusivamente a las tecnologías de la información. En consecuencia la posibilidad de adquirir las competencias necesarias para desempeñarme como ingeniero en algún área de la inteligencia artificial fue un incentivo adicional a la hora de iniciar este proyecto.

## 1.2 Descripción general

El presente trabajo se desarrolla en una fábrica automotriz en el marco laboral de pasante en el departamento de Calidad Fabricación.

Uno de los enfoques de la industria automovilística para mejorar la calidad de sus productos consiste en disminuir la cantidad de fallas por vehículo. Para lograr este objetivo se recopilan los defectos de cada vehículo para realizar diversos análisis y así aplicar estrategias para reducir el impacto de los mismos en la producción. No obstante estos procesos cuya meta es reducir la cantidad de defectos presentan serias dificultades lo que impide obtener el máximo provecho de estos datos. Se consensuó con la compañía realizar un análisis con la finalidad de obtener valor transformando estos datos en información, lo que permitirá tomar acciones anticipadas para mitigar los defectos o su tiempo de reparación.

Debido a las tareas realizadas en la jornada laboral, se poseía conocimiento de la naturaleza de los datos. Por tanto se planteó el diseño de un sistema de inteligencia artificial cuyo fin es obtener en tiempo real la cantidad de defectos graves. Los beneficios que surgen de esta propuesta son tangibles en ahorros para la compañía ya que permite mejorar la planificación en diversos procesos.

Diseñar este tipo de sistema nos permite agrupar los procesos que lo componen en 3 áreas (figura 1):

- Ingeniería de datos.
- Ciencia de datos.
- Ingeniería de *Machine-Learning* ó *ML-Ops*.

Cada una de estas áreas abarca diversos procesos, los cuales se irán desarrollando en diferentes secciones del documento (indicados por los superíndices en la figura 1):

- **Limpieza de datos<sup>(1)</sup>**: sección 3.2.2
- **Preprocesamiento de datos<sup>(2)</sup>**: sección 3.2.3.
- **Análisis y visualización<sup>(3)</sup>**: sección 3.2.4.
- **Imputación de datos<sup>(4)</sup>**: sección 3.2.5.
- **Formateo de datos<sup>(5)</sup>**: sección 3.2.6.
- **Entrenamiento del modelo<sup>(6)</sup>**: sección 4.
- **Despliegue del modelo<sup>(7)</sup>**: sección 5.

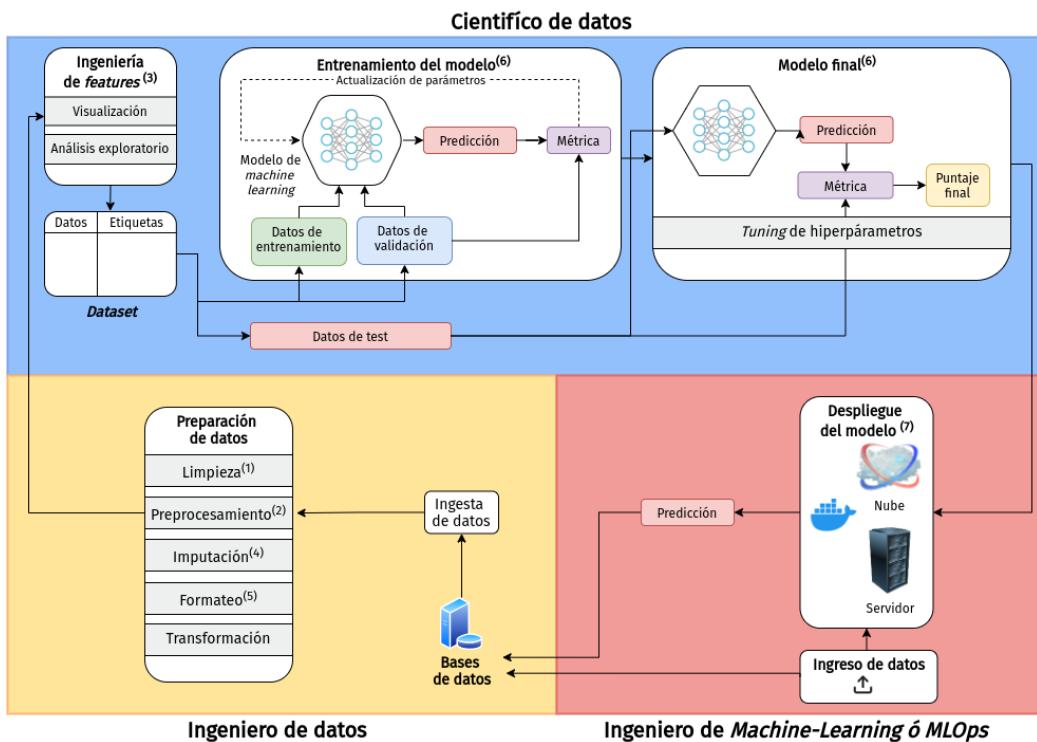


Figura 1: Arquitectura de un sistema de inteligencia artificial.

## 1.3 Objetivos

### 1.3.1 Objetivos principales

Aplicar un modelo de *Machine Learning* cuyo fin será el pronóstico de la cantidad de defectos graves de las unidades en la línea de producción de una planta automotriz.

### 1.3.2 Objetivos secundarios

- Evaluar los diversos algoritmos disponibles de *Machine Learning*, explorando todas las posibilidades en aras de identificar el modelo que mejor se adapte al problema a afrontar.
- Establecer y evaluar la mejor configuración de parámetros para una determinada arquitectura de red con el fin de optimizar su desempeño.
- Analizar la entropía de la serie de datos para determinar si es posible establecer una correlación.
- Examinar el problema a afrontar en su contexto, para así determinar los requerimientos del mismo. Posteriormente haciendo uso de las herramientas para manipulación de datos obtener un formato acorde de la serie temporal para alimentar los modelos.

## 1.4 Requerimientos

En ingeniería, los requerimientos se utilizan como datos de entrada en la etapa del diseño del producto. Establecen qué debe hacer el sistema, aunque no especifican la manera en que debe hacerlo. [1]

Existen dos tipos de requerimientos:

- **Funcionales:** son enunciados acerca de servicios que el sistema debe brindar, su reacción a entradas particulares y comportamiento en situaciones particulares.
- **No funcionales:** no se refieren directamente a la funcionalidad del sistema sino a las propiedades que emergen de ella como la fiabilidad, la capacidad de almacenamiento, disponibilidad, etc. Suelen aplicarse al sistema como un todo, más que a características específicas del sistema.

Para especificar estos requerimientos se ejecutaron las siguientes etapas:

- **Entrevista:** Se realizó una entrevista con el responsable del departamento Calidad Fabricación en la cual se planteó la posibilidad de crear este sistema dado el potencial de los datos recabados por la usina. Esta propuesta fue aceptada y se elevó a los superiores para su aprobación debido a materia de confidencialidad.
- **Investigación del estado del arte:** Se profundizó en el conocimiento sobre las herramientas del campo de la ciencia de datos y *machine-learning*. Esto incluyó el perfeccionamiento del uso de las librerías e investigación sobre algoritmos y modelos de inteligencia artificial, lo que permitió obtener una mejor perspectiva sobre el sistema a desarrollar al conocer las posibilidades y limitaciones del área.
- **Entrevista cerrada:** Se repitió la primer etapa pero teniendo en cuenta la información recabada en la etapa anterior. Se transmitió el conocimiento adquirido al responsable para alinear los objetivos propuestos, añadiendo nuevas consideraciones y descartando funcionalidades inviables.

### 1.4.1 Requerimientos funcionales

Los requerimientos del sistema han sido obtenidos en base al estudio teórico de los sistemas disponibles para lograr nuestro objetivo. Los mismos son presentados en el Cuadro 1.

ID	Descripción
RF1	El sistema debe tomar los datos desde las bases de datos de la compañía.
RF2	El <i>pipeline</i> de datos debe procesar los datos para entregarlos al modelo en un formato acorde.
RF3	El <i>pipeline</i> de datos debe tolerar fallos en la entrada de datos soportando formatos imprevistos.
RF4	El modelo debe guardar todas sus inferencias en la base de datos para revisión de rendimiento.
RF5	El modelo debe recibir y entregar un flujo continuo de datos en tiempo real.
RF6	El sistema debe permitir un reporte que se crea dentro del <i>pipeline</i> .
RF7	El sistema debe proveer una API para ser consumida desde una página web.

Cuadro 1: Requerimientos funcionales.

#### 1.4.2 Requerimientos no funcionales

Estos fueron especificados en conjunto con el director del Proyecto Integrador y acordados en la entrevista final con el responsable de la fábrica.

ID	Descripción
RNF1	Optimizar el tiempo de respuesta del <i>pipeline</i> para evitar cuello de botella.
RNF2	Diseñar pruebas para corroborar fiabilidad del sistema.
RNF3	El sistema será desarrollado enteramente en un lenguaje que facilite la integración entre los diversos componentes del sistema.
RNF4	El modelo deberá lograr un rendimiento satisfactorio mínimo ( $65\% \leq$ precisión).
RNF5	El sistema debe ser accesible para cualquier usuario en la fábrica que lo necesite.
RNF6	El sistema utilizará un <i>framework</i> de <i>machine-learning</i> con respaldo de la comunidad para obtener soporte.
RNF7	El sistema debe proveer una API para ser consumida desde una página web.

Cuadro 2: Requerimientos no funcionales.

## 1.5 Riesgos

Un riesgo se define como un evento o condición incierta que en caso de ocurrir tendrá consecuencia sobre al menos uno de los requerimientos del proyecto.

### 1.5.1 Tipos de riesgos

Se pueden identificar varios tipos de riesgos en un sistema como el que se intenta desarrollar.

- **Riesgos de proyecto (P)**: amenazan la planificación del proyecto, principalmente los aspectos relacionados al tiempo.
- **Riesgos técnicos (T)**: repercuten directamente en la calidad del producto.
- **Riesgos de negocio (N)**: comprometen la viabilidad del proyecto.

## 1.6 Gestión de riesgos

Es un enfoque estructurado para *manejar* la incertidumbre ante amenazas a través de una secuencia de actividades que tienen como objetivo reducir los efectos negativos de los riesgos ya sea evadiéndolos si es posible o aceptando las consecuencias en el peor de los casos.

Las etapas que lo componen son las siguientes:

1. **Identificación de riesgos.**
2. **Análisis de riesgos.**
3. **Selección de riesgos.**
4. **Elaboración de planes de riesgos.**

### 1.6.1 Identificación de los riesgos

Se realizó en base al conocimiento existente sobre los datos y el modelo a desarrollar. Los riesgos fueron clasificados según lo visto en la sección 1.5.1.

Riesgo	Descripción
Aumento del tiempo de desarrollo del proyecto.	Que el producto a realizar sea de mayor magnitud a la estimada, llevando a mayores tiempos de desarrollo.
Planificación optimista.	Se planifica el proyecto en base a criterios de situaciones óptimas o ideales en las que no se presentan problemas graves.

Cuadro 3: Identificación de riesgos de proyecto.

Riesgo	Descripción
Desestimación de inversión.	La compañía una vez que el desarrollo del producto se encuentre en etapa avanzada hará un análisis de costes y beneficios para decidir la culminación total del proyecto.
Falta de personal técnico para conformar equipo de desarrollo.	La compañía debe evaluar el despliegue técnico que necesita el modelo ya que la naturaleza de este es transversal a varios sistemas de la fábrica y su implementación requiere de un equipo de desarrollo especializado.

Cuadro 4: Identificación de riesgos de negocio.

Riesgo	Descripción
Arquitectura de sistemas de la organización no apta para el despliegue del modelo.	Muchos de los sistemas utilizados en la organización datan de largo tiempo e inclusive no reciben mantenimiento activo. Por tanto es necesario revisar la factibilidad de la integración del producto con estos sistemas y la obtención de los datos necesarios.
Servidor con <i>hardware</i> insuficiente para ejecutar el modelo.	La compañía deberá evaluar la adquisición o alquiler de <i>hardware</i> específico y necesario para el entrenamiento e inferencia del modelo.
Incapacidad de acceso en tiempo real a las bases de datos.	Se deben actualizar las bases de datos actuales (actualmente bajo el modelo SAP) para proveer los datos al modelo en tiempo real para su inferencia.
Pérdida del modelo entrenado por situación imprevista	En ciertas ocasiones los modelos son entrenados por largos intervalos de tiempo para luego obtener los reportes y sus pesos. Sin embargo a veces surgen situaciones imprevistas que corrompen el código con el tiempo de ejecución avanzado provocando la pérdida de estos datos.

Cuadro 5: Identificación de riesgos técnicos.

### 1.6.2 Análisis de riesgos

No es posible tratar todos los riesgos existentes por tanto se procederá a cuantificar los mismos para compararlos y priorizar aquellos que se consideren con mayor relevancia para su tratamiento.

La posibilidad de ocurrencia se medirá de forma continua entre 0 y 1, donde 0 significa que es prácticamente imposible que ocurra y 1 es muy probable. Por otro lado, el impacto se medirá en una escala discreta entre 1 y 4 cuyos niveles representan:

1. Insignificante.
2. Tolerable.
3. Importante.
4. Muy grave.

La exposición ( $E$ ) es el producto entre la probabilidad ( $P$ ) y el impacto del riesgo ( $R$ ), representando con este valor la gravedad del mismo.

$$E = PI$$

Riesgos	Tipo	P	I	E
Aumento de la complejidad del proyecto.	P	0,4	2	0,8
Planificación optimista.	P	0,5	3	1,5
Desestimación de inversión.	N	0,8	3	2,4
Falta de personal técnico para conformar equipo de desarrollo.	N	0,6	3	1,8
Arquitectura de sistemas de la organización no apta para el despliegue del modelo.	T	0,9	3	<b>2,7</b>
Servidor con hardware insuficiente para ejecutar el modelo.	T	0,7	3	2,1
Se debe acceder a las bases de datos a través de un protocolo que permite la transferencia de datos en tiempo real.	T	0,7	4	<b>2,8</b>
Pérdida del modelo entrenado por situación imprevista	T	0,6	2	1,2

Cuadro 6: Riesgos.

### 1.6.3 Selección de riesgos

Para esta tarea se decidió utilizar el principio de Pareto [2], el cual establece que de forma general, el 20% del esfuerzo produce el 80% de los resultados. O dicho de otra manera: el 80% de las consecuencias provienen del 20% de las causas, así le daremos prioridad al conjunto de riesgos de mayor exposición para mitigar sus efectos. El 20% de los 8 riesgos mostrados en el cuadro 6 es 1.6, el cual redondearemos a 2.

### 1.6.4 Elaboración de planes de riesgos

Para esta tarea realizaremos dos tipos de planes.

- **Plan de mitigación:** su objetivo es reducir la probabilidad de ocurrencia del riesgo o el impacto que el mismo puede provocar.
- **Plan de contingencia:** está comprendido por acciones que se deberán realizar solamente si el riesgo se presenta.

Los dos riesgos principales son de tipo técnico, esto en gran parte se debe a la escasa actualización de los sistemas de la empresa. Esto provoca un gran déficit tecnológico que vulnera las oportunidades de soluciones informáticas en diversas áreas de la fábrica. No obstante, en el cuadro 7 definimos planes de mitigación y contingencia para cada uno de estos riesgos.

<b>Riesgo</b>	<b>Plan de mitigación</b>	<b>Plan de contingencia</b>
Arquitectura de sistemas de la organización no apta para el despliegue del modelo.	Contactarse desde una etapa temprana con el departamento de informática de la empresa para determinar el estado actual de los sistemas. En caso que sea realizable la implementación podemos trabajar desde etapas tempranas teniendo en cuenta los sistemas, de este modo se ahorran tiempos al momento de la integración.	Realizar un reporte detallado de costos y beneficios del proyecto para que los directivos analicen de forma más organizada la información para tomar la decisión. Esto incluiría diversos beneficios como la actualización de varios sistemas de la fábrica que no reciben soporte ya y por tanto tienen un rendimiento reducido además de presentar vulnerabilidades.
Incapacidad de acceso en tiempo real a las bases de datos.	Comunicarse con el departamento de informática con el objetivo de obtener soporte oficial de la empresa creadora del sistema de base de datos. La idea es explorar la posibilidad de crear un protocolo que permita conectar nuestro sistema con la base de datos en tiempo real.	Proponer la migración o duplicación de los datos necesarios a una base de datos relacional que permita realizar consultas en tiempo real.

Cuadro 7: Elaboración de planes de riesgo.

## 2 Inteligencia Artificial

La *Inteligencia Artificial (Artificial Intelligence)* se define como el estudio de los "agentes inteligentes", i.e. cualquier dispositivo que perciba su entorno y tome medidas que maximicen sus posibilidades de lograr con éxito sus objetivos.

Poole et al. [3]

Esta definición nos da la idea de que la IA es un sistema reactivo, que reacciona a cambios externos y actúa en consecuencia.

### 2.1 Aprendizaje automático

El *aprendizaje automático (Machine Learning)* es el estudio científico de algoritmos y modelos estadísticos que los sistemas informáticos utilizan para realizar una tarea específica sin utilizar instrucciones explícitas, sino que se basan en patrones e inferencia. Es visto como un subcampo de inteligencia artificial. Los algoritmos de aprendizaje automático crean un modelo matemático basado en datos de muestra, conocidos como "datos de entrenamiento", para hacer predicciones o decisiones sin ser programado explícitamente para realizar la tarea.

Bishop [4]

Es importante destacar la independencia del aprendizaje automático al momento de tomar decisiones a partir de los datos proporcionados sin intervención externa, es decir que no hay una especificación de reglas que dictan cómo deben ser tomadas estas decisiones. A su vez, los modelos obtenidos a partir de los algoritmos de *Machine Learning* deben tener la capacidad de predecir a partir de nuevos datos, nunca antes procesados por el modelo, a esto se lo conoce como **generalización**.

En el anexo del proyecto integrador se ha realizado un profundo análisis de los métodos y algoritmos más utilizados en el campo de la Inteligencia Artificial.

### 3 Procesamiento de datos

#### 3.1 Origen de los datos

Los datos recabados provienen de la línea de fabricación de vehículos utilitarios en una fábrica de automotores, en la cual se utiliza un sistema de diversas auditorias para identificar defectos.

##### 3.1.1 Organización de la usina

La fábrica está compuesta de departamentos, talleres y unidades de trabajo en ese orden jerárquico. A continuación listaremos los departamentos:

- Calidad (CALI)
- Embutición (EMBU)
- Ingeniería (DLI)
- Logística (SQF)
- Montaje (MONT)
- Pintura (PINT)
- Soldadura (SOLD)
- División de abastecimiento de piezas (DIVD)

La línea de fabricación consta de 4 departamentos que intervienen directamente en el ensamblaje del vehículo:

EMBU → SOLD → PINT → MONT

Los demás departamentos son una parte vital de la producción pero intervienen indirectamente, y CALI se encarga de verificar los defectos con mayor recurrencia de los departamentos. La figura 2 nos esclarece esta idea.

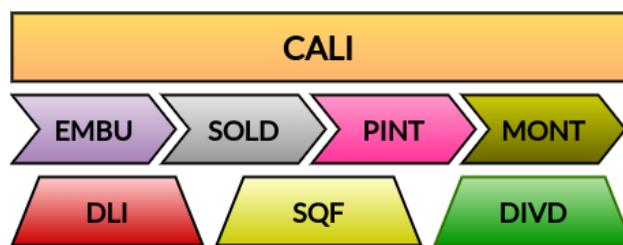


Figura 2: Diagrama de departamentos de la fábrica.

Cabe destacar a su vez que cada departamento posee talleres, que a su vez poseen unidades de trabajo obteniendo una estructura jerárquica de tipo árbol.

### 3.1.2 Nomenclatura de defectos

Un defecto (DEF) en su esencia se compone de un elemento (ELE), un incidente (INC) y opcionalmente una localización (LOC).

$$\text{DEF} = \text{ELE} + \text{INC} + \text{LOC}$$

Tanto ELE como INC están definido inequívocamente por un código de 4 caracteres mientras que LOC puede variar, lo que lleva a que un defecto posea al menos 8 caracteres.

### 3.1.3 Gravedad

Además los defectos se categorizan según la gravedad (GVD) del mismo en:

- V1: graves.
- V2: leves.
- V3: imperceptibles (para nuestro análisis serán descartados).

### 3.1.4 Tipo

A su vez los defectos están categorizados por familia o tipo de defectos:

- APR: aprietes.
- ASP: aspecto.
- DGRC: degradaciones.
- ELEC: eléctrico.
- ESTQ: estanqueidad.
- FCIO: funcionamiento.
- FLDS: fluidos.
- FTES: faltantes.
- GMTR: geometría.
- MOP: modo operatorio.
- NCON: no conforme.
- RDOS: ruidos.

### 3.1.5 Puntos de reconocimiento de defectos

Los defectos son detectados en diversos puntos de captaje que pueden definir (o no) a que departamento o taller pertenece el defecto con un doble objetivo. En primer lugar para que los vehículos que ya poseen el defecto detectado sean derivados a puntos de retoque para ser reparados. Y en segundo para analizar y ejecutar diversas estrategias con el fin atacar el problema raíz para así erradicar el defecto.

Los defectos detectados provienen de las siguientes fuentes de datos:

- **Defectos por unidad** (DPU): los defectos encontrados en los puntos de reconocimiento a lo largo de toda la línea de producción.
- **Carrocería pintura-soldadura** (CAPS): se realiza una auditoria al final de la línea de pintura sobre una muestra de carrocerías.
- **Plan estático-dinámico** (PESD): una vez finalizado el proceso de fabricación del vehículo, se le realizan pruebas de estanqueidad, prueba de manejo, entre otros con objeto de encontrar defectos que en línea no sería posible.
- **SAVES**: Auditoria de 5 minutos de una muestra aleatoria de vehículos en línea final.

Sin embargo solo obtenemos datos a partir de los puntos de captaje de **SOLD** dado que en **EMBU** las partes aún no poseen un identificador único del vehículo.

### 3.2 Ingeniería de datos

Es vital realizar el procesamiento de los datos para alimentar el modelo con los datos en un formato acorde, lo que se conoce en la industria como **ingeniería de datos**.

Para ello se ha dividido la tarea en las etapas o *stages* que se muestran en la figura 3.



Figura 3: Procesamiento de datos segmentado en etapas.

En el campo de ciencia de datos la comunidad ha consensuado de facto que el estándar es utilizar como entorno de trabajo **Jupyter-Notebook**, que soporta **Python** como lenguaje de programación. Existen otros lenguajes reconocidos como **R** pero poseen mayor difusión en comunidades académicas.

Las librerías que serán parte del proyecto son las siguientes:

- **Pandas**: manejo de tablas y datos.
- **Numpy**: operaciones matriciales.
- **Matplotlib, Seaborn**: visualización de datos.

### 3.2.1 Recolección de datos

Etapa de búsqueda y recolección de los datos que alimentarán el modelo. En nuestro caso se dividió en recolectar en la organización los datos de defectos, que fueron obtenidos desde 4 auditorías diferentes que fueron presentadas en 3.1.5.

### 3.2.2 Limpieza de datos

Consiste en inspeccionar los datos obtenidos en búsqueda de posibles errores o datos que quizás es posible obtener pero por diversas razones se han perdido y debemos intentar recuperar.

Ambos casos sucedieron, por tanto se procedió a recuperar los datos cuando fue posible, y en los que no se procedió al descarte debido a que no eran útiles para la tarea.

### 3.2.3 Preprocesamiento de datos

En este punto es donde debemos prestar atención a los detalles al inspeccionar nuestro *dataset*.

En primer lugar se realizó una homogeneización de los datos, *i.e.* quizás el mismo dato se puede estar refiriendo a lo mismo pero tienen *tags* diferentes, por tanto es necesario revisar los datos columna por columna. Esto sucedió debido a que provenir los datos de diversas fuentes cada una tenía una forma de nombrarlos.

Posteriormente fue necesaria la detección de incongruencias, advirtiendo inconvenientes tales como departamentos inexistentes. Además todos los defectos que no hayan sido catalogados en el campo GVD pasaron a ser V2.

Debido a la estructura jerárquica de la usina, si tenemos definido la UET a la que pertenece el defecto, escalando hacia arriba podemos obtener el taller y el departamento correspondiente. Esta operación se realizó para todo el *dataset*.

Una vez realizadas estas operaciones *dataset* se encuentra en condiciones para su análisis.

### 3.2.4 Análisis y visualización de datos

Para obtener una perspectiva más amplia del problema a tratar es conveniente analizar los datos disponibles. El *dataset* se compone de 229934 filas que cada una representa un defecto y 17 columnas.

Algunas estadísticas *grosso modo*, que podemos obtener:

- 10780 vehículos fueron fabricados durante 2020.
- 5224 defectos diferentes fueron registrados en el sistema.
- Se registran en promedio 19,6 defectos por vehículo.

Si verificamos el origen de los datos nos percataremos que casi la totalidad de ellos provienen de DPU, disputándose el pequeño margen restante PESD, CAPS y SAVES.

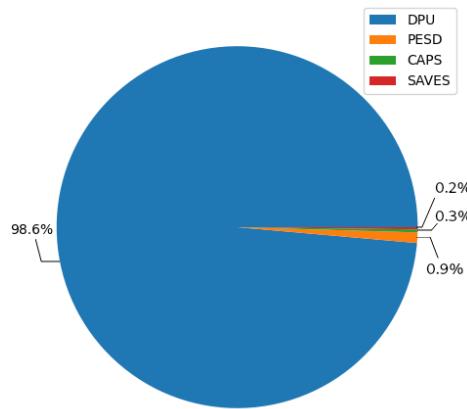


Figura 4: Distribución de los defectos según AUDI.

No obstante, no se debe perder el propósito el cual es el pronostico de la cantidad de defectos graves que ocurrirán en función a todos los defectos registrados con anterioridad. En la fig. 6 observamos que la gran mayoría de defectos registrados son V2 lo que podría ser útil para nuestro modelo.

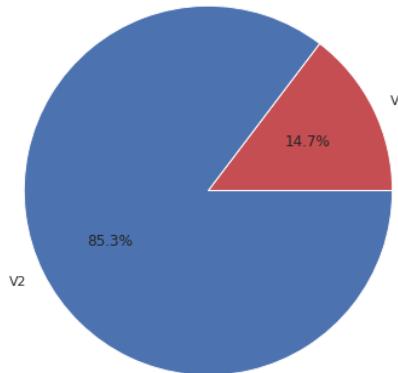


Figura 5: Distribución de los defectos según gravedad.

La fig. 6 nos muestra que los defectos principalmente ocurren en 3: PINT, SOLD y MONT, un poco más atrás en la participación está SQF.

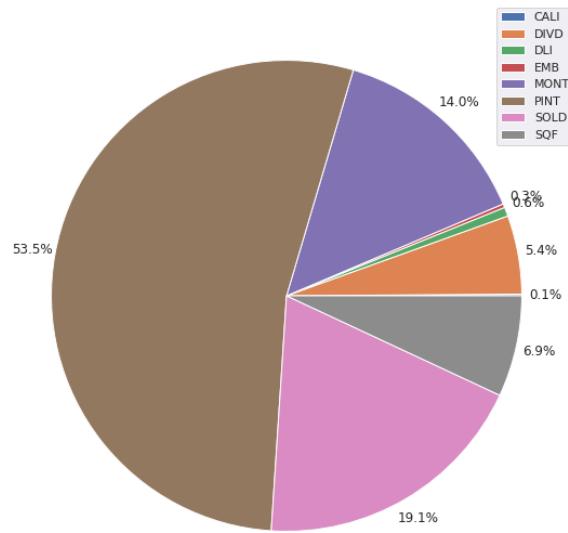


Figura 6: Distribución de los defectos por DPTO.

La fig. 7 resume que casi 7 decimos de los defectos totales son de tipo ASP, seguido muy por detrás por DGRC y en tercer lugar los de MOP.

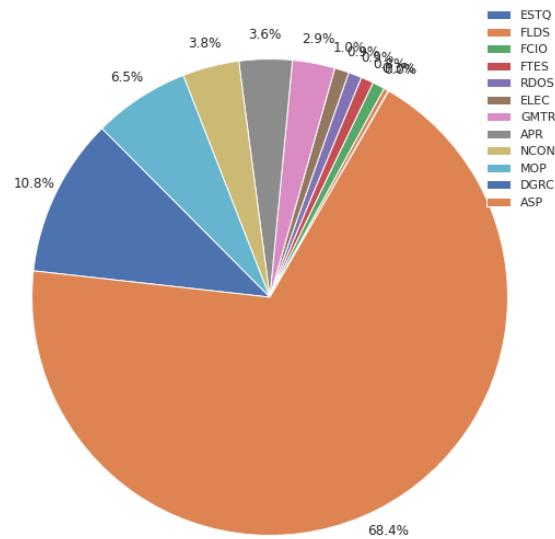


Figura 7: Distribución de los defectos por TIPO.

Mediante un mapa de calor (fig. 8) se cruzaron las características TIPO y DPTO. Advertimos una cierta correlación entre algunos tipos de defectos y los departamentos, a tal punto que en algunos departamentos ni siquiera existen determinados tipos de defectos. Sería beneficioso que el modelo pueda aprender esta característica para maximizar sus posibilidades.



Figura 8: Cantidad de defectos cruzando DPTO y TIPO.

Pese a esto no se debe creer que aunque una combinación tenga una gran cantidad de defectos, la mayoría de estos serán V1. La fig. 9 nos muestra la densidad de V1 sobre el total de defectos, permitiéndonos establecer que determinadas combinaciones de TIPO y DPTO son más probables que sean V1 (incluso algunas poseen valores considerablemente más altos que las demás).

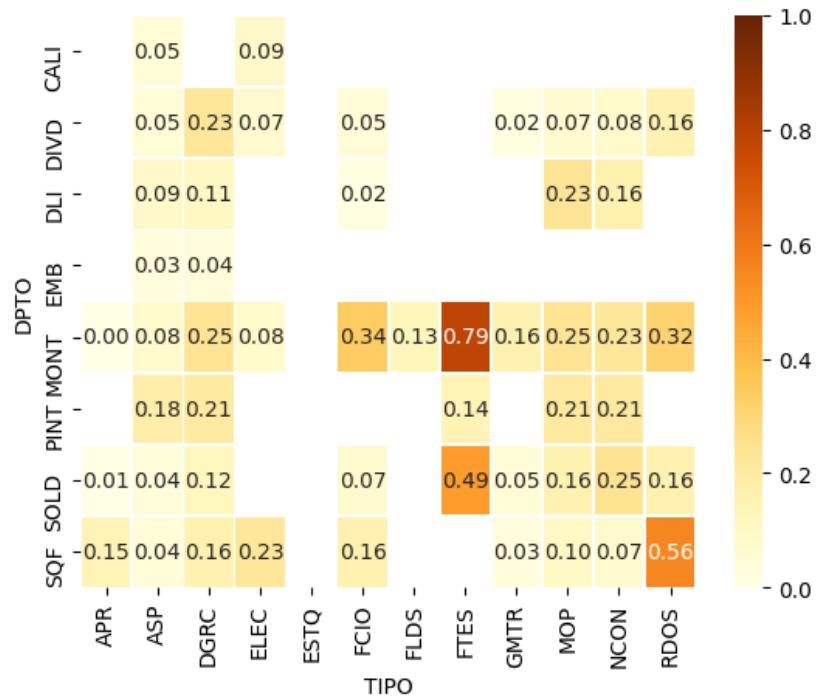


Figura 9: Fracción de V1 por cantidad de defectos.

El análisis visual más revelador es el de la figura 10. El 19 de marzo de 2020 se decretó la cuarentena debido a la pandemia provocada por el virus *SARS-CoV-2*, por tanto no se fabricaron vehículos. Los defectos registrados durante esa fecha y el reinicio de la producción en junio se deben a la recuperación de vehículos por retoques. Por esto para que el flujo de los datos hacia el modelo sea continuo solo se tomará desde junio a diciembre (6 meses).

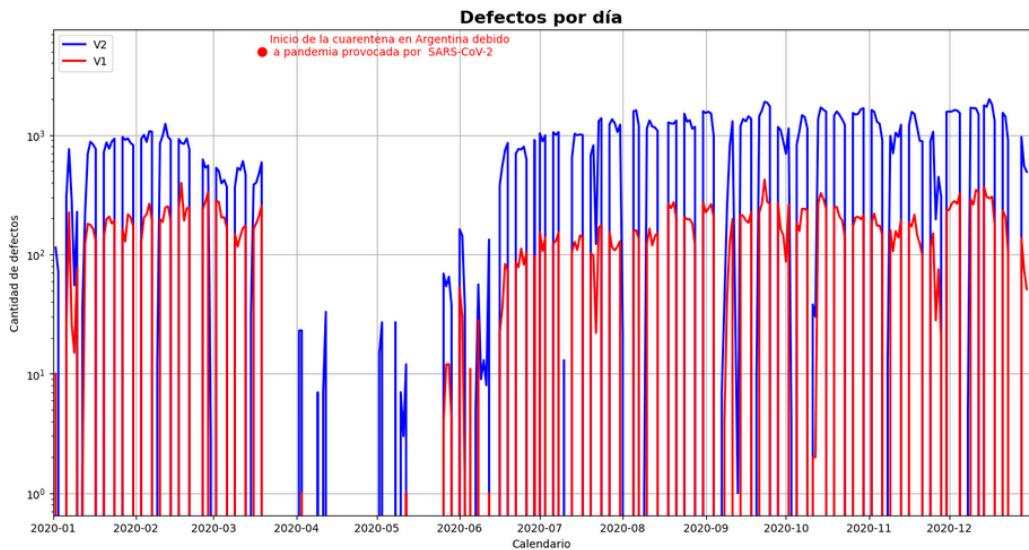


Figura 10: Defectos diarios agrupados por GVD.

Aún así, no podemos notar un patrón específico debido al ruido, por consiguiente incrementamos la ventana de tiempo a una semana (fig. 11). Afortunadamente para nuestro modelo (aunque la escala del eje de ordenadas es logarítmico) la cantidad de defectos V2 sigue la curva de V1 de forma muy similar. Esto es positivo ya que podemos intuir que existe una correlación entre los mismos del cual nuestro modelo podrá obtener provecho.

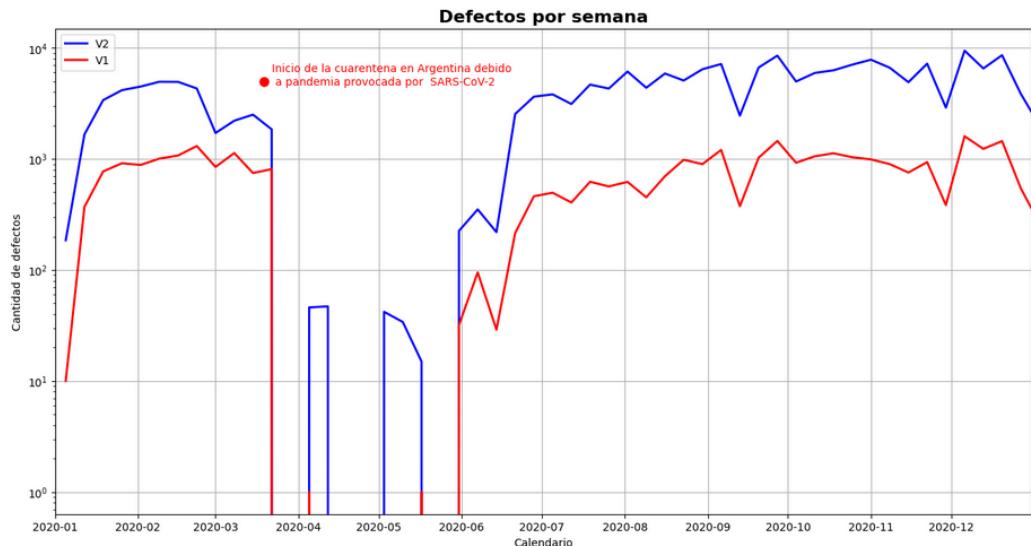


Figura 11: Defectos semanales agrupados por GVD.

Si al gráfico anterior lo desglosamos por auditoria y gravedad (fig. 12) veremos que en las demás auditorias que no sean DPU no hay patrones claros definidos, una de las causas es la baja densidad de datos disponibles.

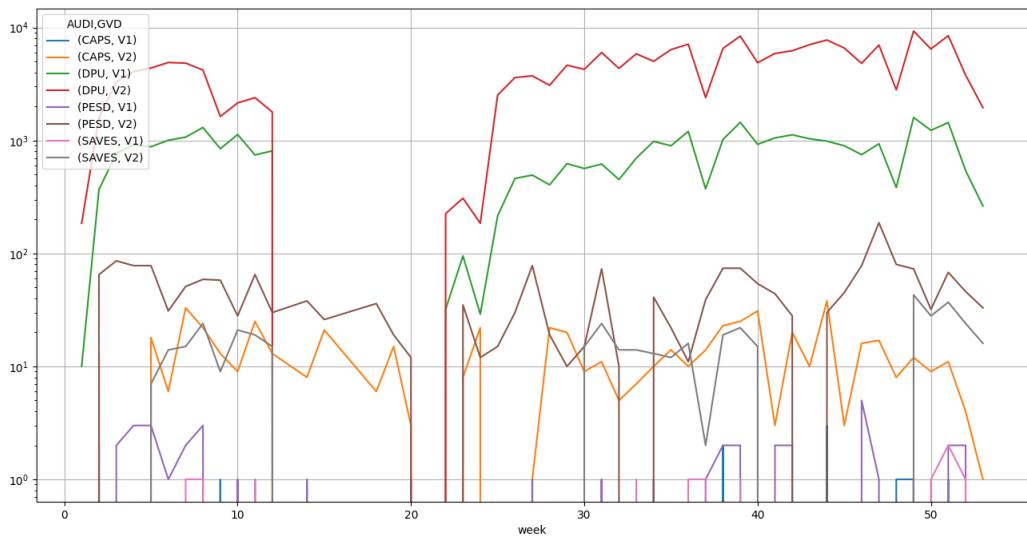


Figura 12: Defectos semanales agrupados por AUDI/GVD.

### 3.2.5 Imputación de datos

En estadística, la imputación es el proceso de reemplazar los datos faltantes (*missing values*) con valores sustituidos. En nuestro *dataset* tenemos 2 *features* con faltantes, una es UET (lo que conlleva a faltantes de DPTO y TALL) y la otra es TIPO.

Como se observa en la fig. 13, los datos faltantes no son demasiados, más precisamente 1052 de UET y 1228 de TIPO. Debido a la gran cantidad de datos disponibles podemos llenar estos valores con algún algoritmo de *Machine Learning* visto con anterioridad.

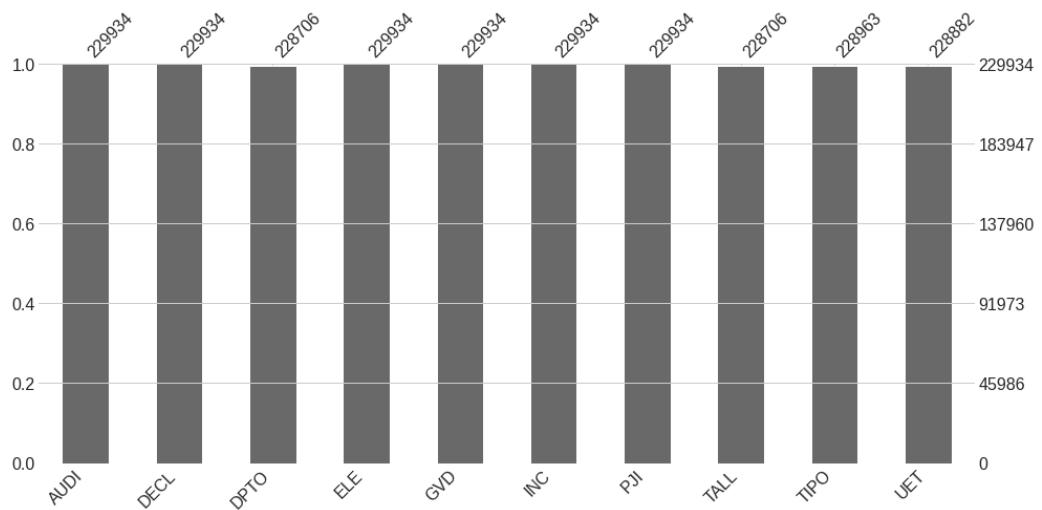


Figura 13: Datos faltantes por *feature*.

### Matriz de correlación

En un *dataset* con muchos atributos, el conjunto de valores de correlación entre pares de sus atributos forma una matriz que se denomina matriz de correlación.

Existen varios métodos para calcular un valor de correlación. El más popular es el coeficiente de correlación de *Pearson*. Sin embargo, debe notarse que mide solo la relación lineal entre dos variables. En otras palabras, es posible que no pueda revelar una relación no lineal. El valor de la correlación de *Pearson* varía de  $-1$  a  $+1$ , donde  $\pm 1$  describe una correlación positiva/- negativa perfecta y  $0$  significa que no hay correlación. [46]

La matriz de correlación es una matriz simétrica con todos los elementos diagonales iguales a +1. Nos gustaría enfatizar que una matriz de correlación solo brinda información sobre la correlación y NO es una herramienta plenamente confiable para estudiar la causalidad.

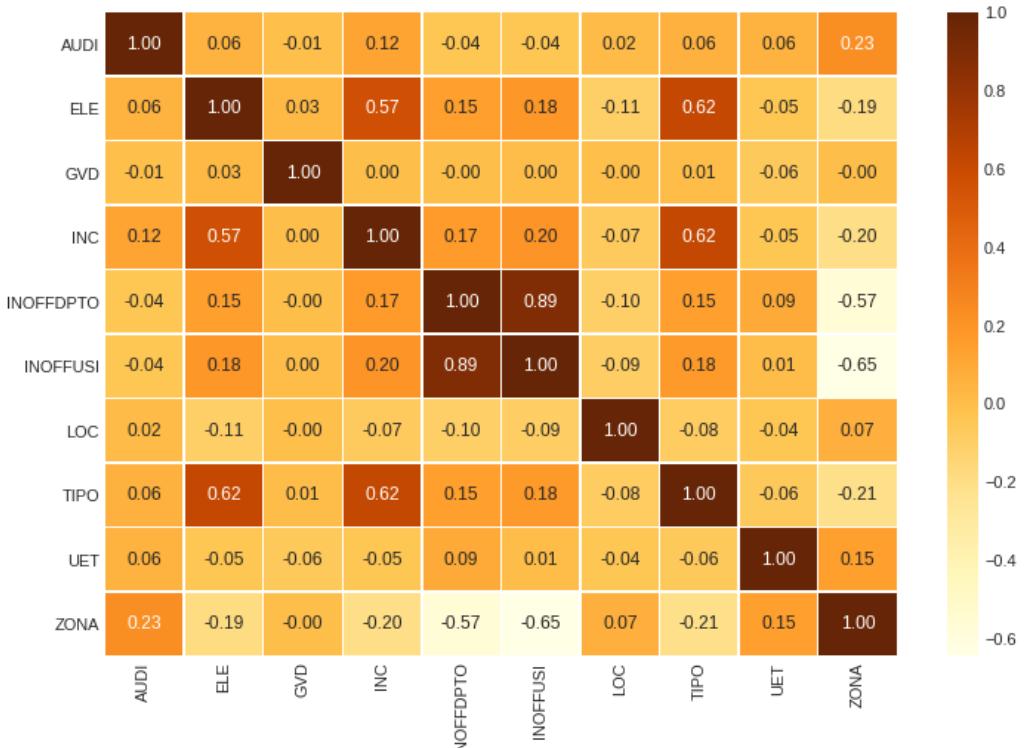


Figura 14: Matriz de correlación de nuestro *dataset*.

En la fig. 14 pondremos atención sobre las dos *features* que nos propusimos predecir.

- **TIPO:** en este caso notamos que las features ELE y INC, tienen una relación notable con TIPO, por tanto esas serán utilizadas y el resto descartadas.
- **UET:** aquí las relaciones están mucho más difusas, por tanto usaremos *features* que estén en las filas a las cuales queremos predecir su UET. Por tanto se seleccionó ELE, INC, LOC, GVD y TIPO.

### ***Scikit-learn***

Dado que el proceso de imputación no es crítico, se optó por utilizar la librería *sk-learn* (en lugar de redes neuronales) que nos provee de algoritmos ya implementados. Además nos permite a través de *pipelines* realizar un preprocesamiento de los datos para posteriormente entrenar al modelo.

Luego de revisar los valores faltantes se decidió utilizar como estimador diversos tipos de clasificadores para así seleccionar el de mejor rendimiento, para posteriormente llenar los valores.

Los siguientes clasificadores fueron seleccionados:

- *KNeighborsClassifier* (KNC)
- *SGDClassifier* (SGDC)
- *RidgeClassifier* (RC)
- *LogisticRegression* (LR)
- *XGBClassifier* (XGBC)
- *DecisionTreeClassifier* (DTC)
- *RandomForestClassifier* (RFC)
- *BaggingClassifier* (BC)

### **Estimación de la precisión del modelo**

Para estimar la precisión de los modelos seleccionados se utilizó *K-fold Cross-Validation* (KFCV) [47].

Una iteración de KFCV se realiza de la siguiente manera:

1. Se genera una permutación aleatoria del conjunto de muestra y se divide en  $K$  subconjuntos (pliegues o *folds*) de aproximadamente el mismo tamaño.
2. De esos  $K$  subconjuntos, un solo subconjunto se retiene como datos de validación para probar el modelo (este subconjunto se llama *testset*), y los restantes  $K - 1$  subconjuntos juntos se utilizan como datos de entrenamiento (*trainset*).
3. Luego, se entrena un modelo en el *trainset* y se evalúa su precisión en el *testset*.
4. El entrenamiento y la evaluación del modelo se repiten  $K$  veces, y cada uno de los  $K$  subconjuntos se utiliza exactamente una vez como *trainset*.

En la fig. 15 se ejemplifica parte del proceso.

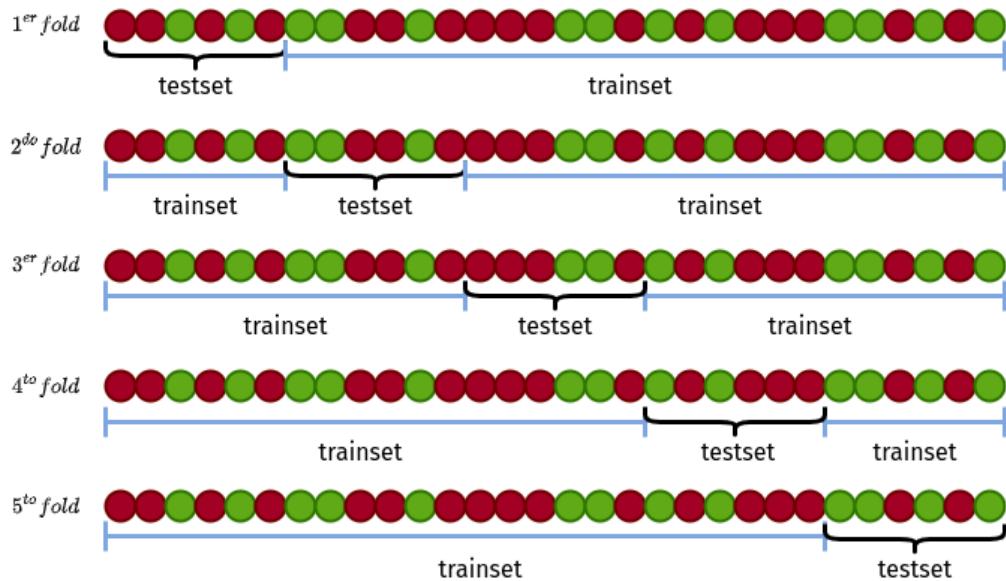


Figura 15: Ejemplo de validación cruzada de 5 veces con 30 muestras.

La estimación de precisión resultante depende de la permutación aleatoria que se generó al comienzo del proceso, ya que afecta la forma en que se partitiona el conjunto de muestras.

Por ello para obtener una estimación más exacta de la precisión, tiene sentido repetir la validación cruzada varias veces y tomar el promedio final después de cada iteración como estimación de precisión resultante.

En nuestro caso debido a que todas las *features* son de tipo categoría usamos la codificación `one-hot encoder`, como observamos en la fig. 16.

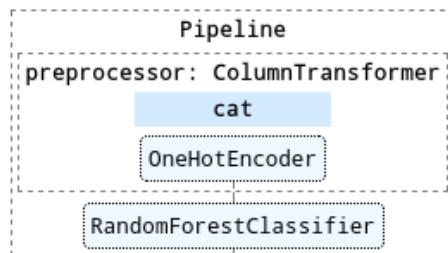


Figura 16: *Pipeline* de *sklearn* para imputar datos.

### Selección de algoritmo

Al momento seleccionar un algoritmo para inferir los valores faltantes debemos tener en cuenta 3 criterios:

- **Tiempo de entrenamiento:** si nuestro *dataset* crece y el poder de cómputo disponible es el mismo, es posible que este valor para un determinado algoritmo sea alto y por consiguiente inutilizable.
- **Tiempo de inferencia:** (también llamado latencia de predicción) es un valor importante ya que como mencionamos en la sección, nuestro modelo necesita un *pipeline* de datos que lo alimente continuamente. Por tanto si el tiempo de inferencia es grande puede provocar un cuello de botella y eventualmente que el *pipeline* se rompa.
- **Rendimiento:** es un valor importante, ya que en la medida que la inferencia de datos sea precisa nuestro modelo funcionará mejor.

### Predicción de UET

Teniendo en cuenta la matriz de correlación de la fig. 14 tomamos en consideración 5 *features*, pero haremos un inspección más minuciosa de las mismas. Utilizando un *miniset* (*i.e.* una fracción de nuestro *dataset*) se fueron probando diferentes *features* para nuestro modelo (debido a que las pruebas requieren menor tiempo de cálculo y se supone que la muestra representa al menos la mayor parte del *dataset*). Tener en cuenta que para que este enfoque funcione es necesario setear el `random_state` de las funciones con una *seed* que es un número entero, el propósito de esto es quitar el componente de azar de los modelos y poder realizar las pruebas sobre los mismos datos.

Los mejores rendimientos se obtenían con ELE, INC y LOC, ergo se descartó GVD y TIPO.

### Cómputo de métricas para cada algoritmo

Algoritmo	Rendimiento [%]	Inferencia [ms]	Entrenamiento [s]
KNC	73,59	31,75	<b>0,59</b>
SGDC	72,56	6,82	14,99
RC	68,49	6,77	34,57
LR	74,20	6,64	89,74
XGBC	72,23	9,82	305,77
DTC	77,22	<b>6,23</b>	4,60
RFC	<b>77,66</b>	20,91	262,14
BC	77,24	9,76	44,56

Cuadro 8: Parámetros de algoritmos para predicción de TIPO

## Rendimiento

En la fig. 17 notamos como RFC, BC y DTC son superiores a los demás algoritmos.

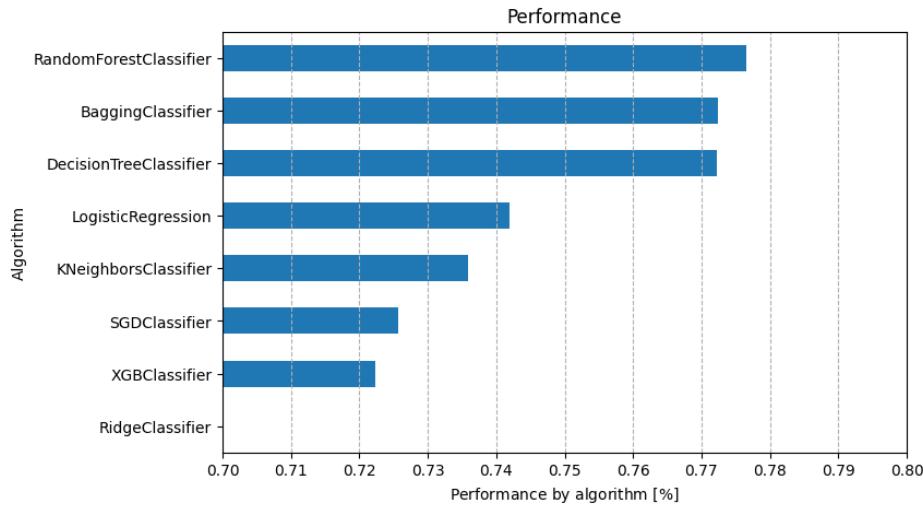


Figura 17: Rendimiento por algoritmo.

## Tiempo de entrenamiento

Hay resultados muy dispares entre los algoritmos, algunos como DTC son muy veloces mientras que otros como RFC o XGBC necesitan demasiado tiempo.

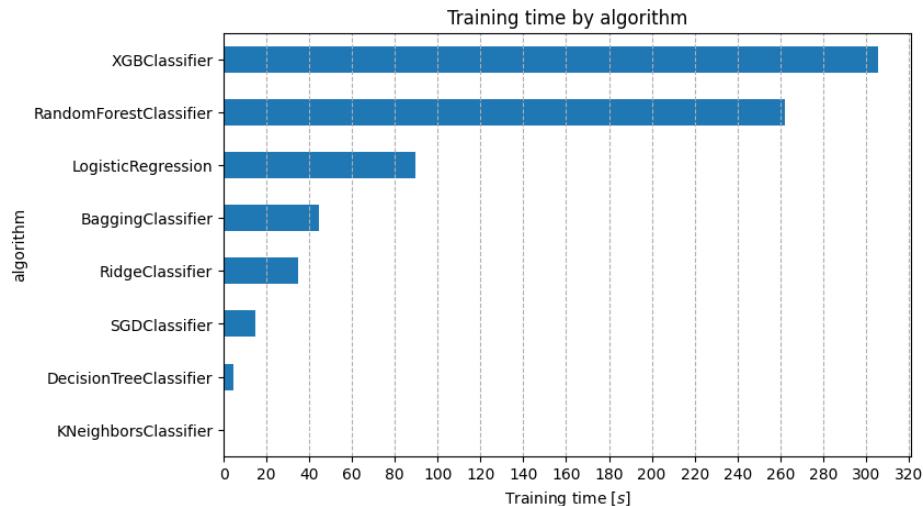


Figura 18: Tiempo de entrenamiento por algoritmo.

### Tiempo de inferencia

Se realizó inferencia atómica (*i.e.* muestra por muestra, en vez de por loteo) sobre 10000 muestras recopilando los tiempos de cada algoritmo. Analicemos el gráfico de cajas expuesto en la fig. 19. La gran mayoría de los algoritmos muestra tiempos cercanos a 5ms, no obstante se pueden observar algunos *outliers* que llegan a 20ms. KNC tiene tiempos muy altos de inferencia sumado a una gran dispersión debido a que el cálculo de la distancia euclídea puede llevar a mucho tiempo de cómputo. RFC por su parte tarda en promedio 5 veces más que los demás algoritmos, lo cual es una desventaja al momento de la decisión final.

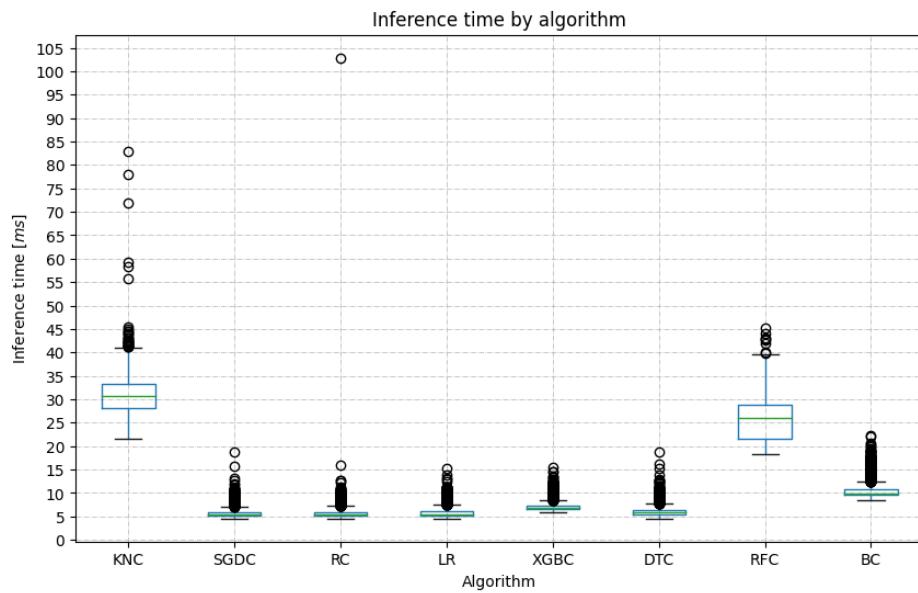


Figura 19: Tiempo de inferencia por algoritmo.

### Comparación visual de métricas

El gráfico de dispersión presentado en la fig. 20 es útil para seleccionar el mejor algoritmo ya que en el eje de abscisas está el rendimiento, en el eje de coordenadas el tiempo de inferencia y el tamaño del círculo se corresponde con el tiempo de entrenamiento. Por tanto el mejor algoritmo será el punto de menor tamaño que se encuentre más cercano a la esquina inferior derecha.

En consecuencia seleccionamos DTC para la imputación de datos de TIPO, ya que a pesar de tener menor rendimiento que RFC tiene mejores tiempos de inferencia y entrenamiento.

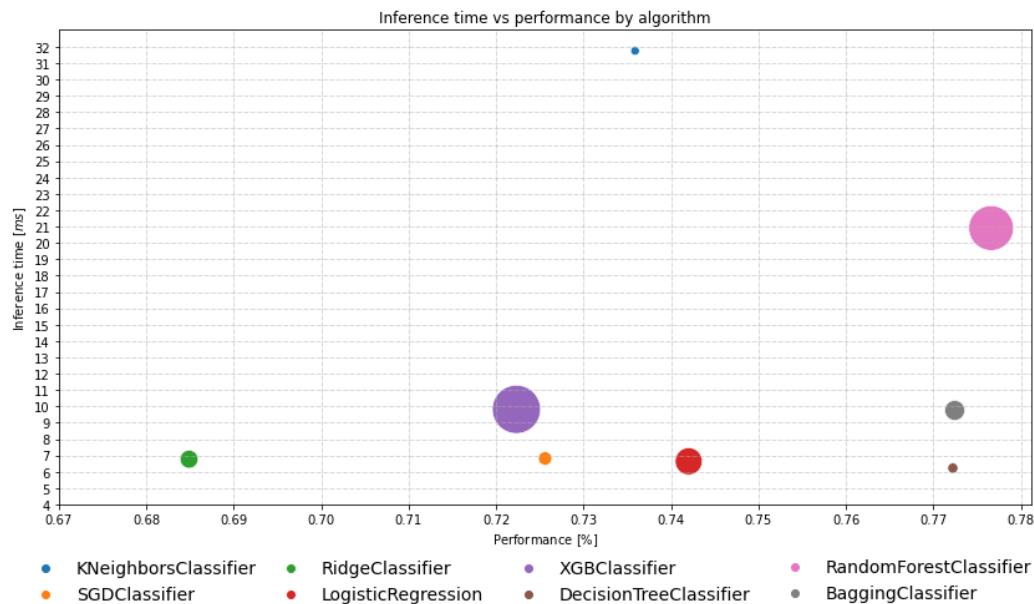


Figura 20: Gráfico de dispersión de métricas por algoritmo.

### Predictión de UET

Se realizará el mismo análisis aplicado a TIPO.

### Cómputo de métricas para cada algoritmo

Algoritmo	Rendimiento [%]	Inferencia [ms]	Entrenamiento [s]
KNC	97,37	28,26	<b>0,74</b>
SGDC	98,95	<b>5,38</b>	3,63
RC	98,21	5,39	7,11
LR	99,06	5,40	29,22
XGBC	<b>99,61</b>	6,44	55,65
DTC	99,02	5,58	4,04
RFC	98,99	19,20	64,65
BC	99,15	9,54	32,03

Cuadro 9: Parámetros de algoritmos para predicción de TIPO. Destacados los valores óptimos por columna.

## Rendimiento

Existe un salto notable entre los rendimientos obtenidos para la predicción de TIPO con respecto a la de UET.

Sugerimos que una razón de esto se debe a la estructura de árbol que posee UET (recordemos que un departamento está compuesto por varios talleres que a su vez poseen múltiples UETs, lo que lleva a una cantidad única de etiquetas superior a la de TIPO). La mayoría de los algoritmos tienen desempeños similares, siendo el óptimo BC.

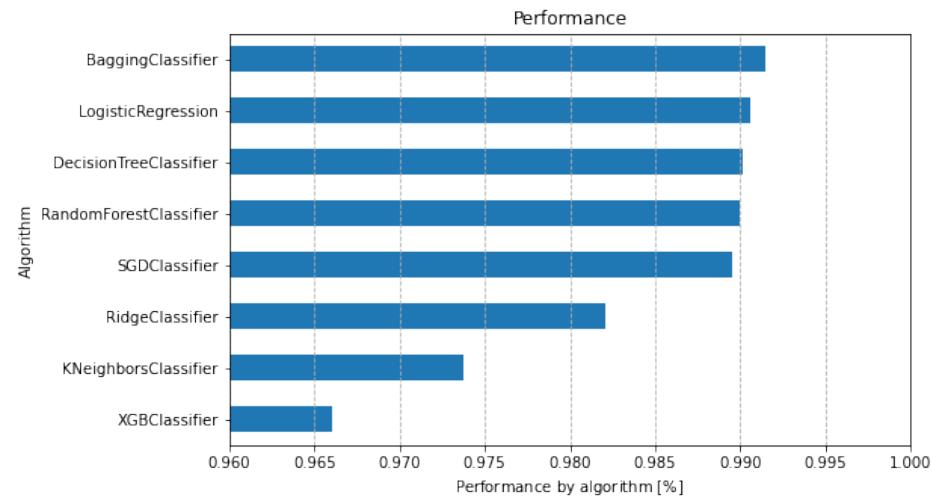


Figura 21: Rendimiento por algoritmo.

## Tiempo de entrenamiento

Al disminuir la complejidad de las etiquetas, esto se ve reflejada en la consecuente disminución del tiempo de entrenamiento de los algoritmos (fig. 22).

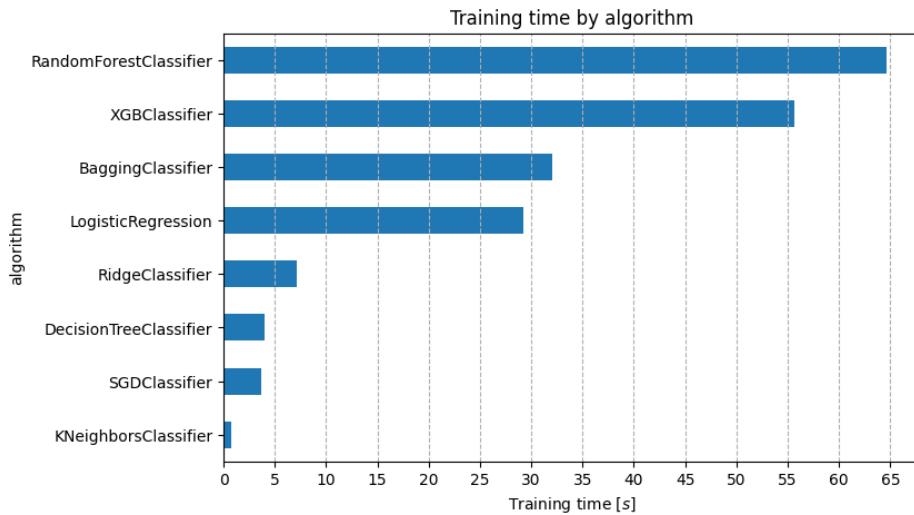


Figura 22: Tiempo de entrenamiento por algoritmo.

### Tiempo de inferencia

En la mayoría de los algoritmos el tiempo de inferencia no supera los  $20ms$ , siendo la media de  $5ms$ . KNC y RFC muestran una gran cantidad de *outliers* y tiempos superiores a la media.

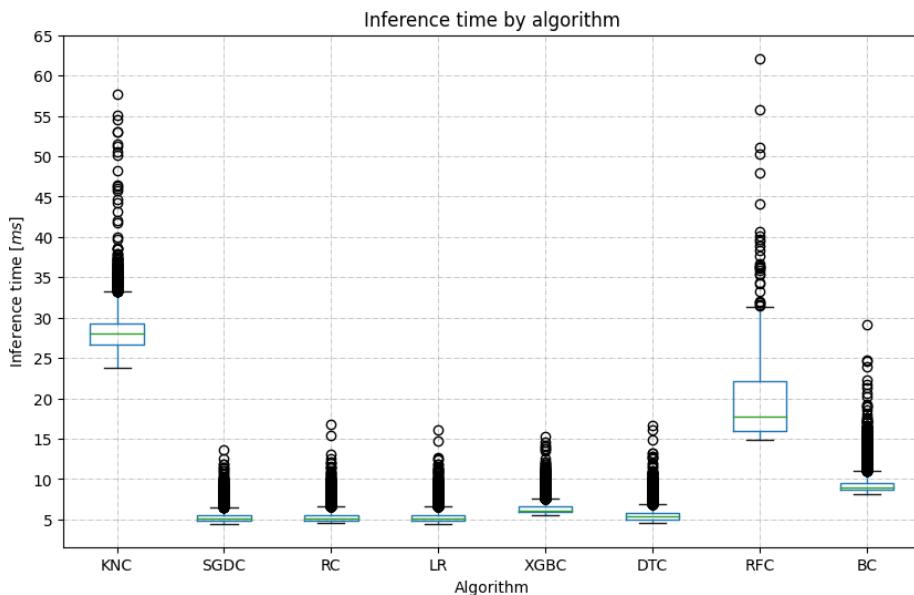


Figura 23: Tiempo de inferencia por algoritmo.

### Comparación visual de métricas

El gráfico de dispersión presentado en la fig. 24 nos permite visualizar que hay dos algoritmos candidatos: LR y DTC. LR exhibe tiempos de inferencia menores y mayor rendimiento, pero un tiempo de entrenamiento superior a DTC. Actualmente el tiempo de entrenamiento es el criterio de menor importancia, por consiguiente LR será el algoritmo para imputar los datos faltantes de TIPO.

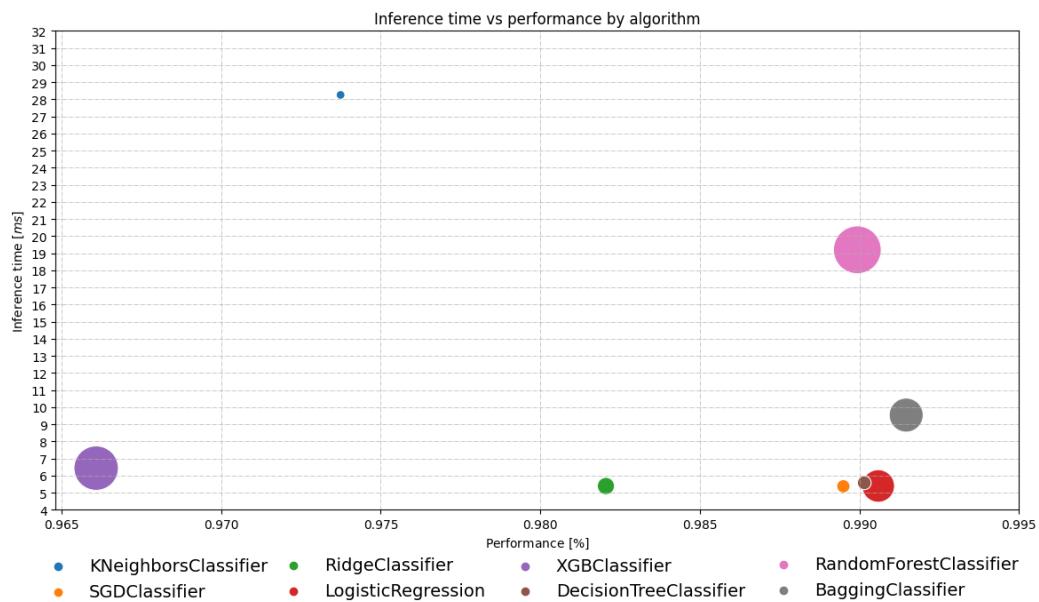


Figura 24: Gráfico de dispersión de métricas por algoritmo.

Para finalizar añadiremos que la fábrica posee personal y procesos específicos para determinar el origen de un defecto (UET) y de a que familia pertenece (TIPO). Implementando algunos de estos algoritmos se podría aliviar su carga de trabajo o reasignar sus horas laborales a tareas de mayor impacto.

#### 3.2.6 Formateo de datos

El paso final antes de alimentar nuestro modelos es obtener rodajas (*slices*) de tiempo con funciones de agregación de tipo `sum`, `count`, entre otras de nuestros datos.

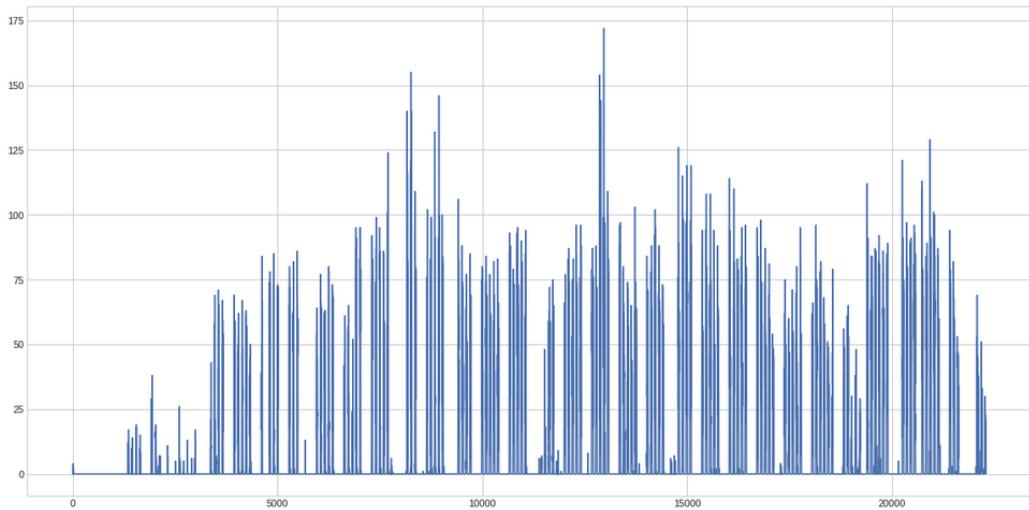


Figura 25: Gráfico temporal de cantidad de defectos.

### Serie temporal univariada

Pero como observamos en la Fig. 25, nuestra serie temporal dista mucho de ser continua. Esto es un inconveniente ya que se ha demostrado que a los modelos les cuesta mucho aprender sobre series temporales intermitentes.

Por esta razón se tomarán las siguientes consideraciones:

- tomar el periodo posterior a la cuarentena por *Sars-CoV-2*.
- quitar los días sin o con baja producción.
- quitar horarios no laborales.

En la Fig. 26 donde aplicamos la función escalón, notamos los huecos que tenemos en nuestra serie, por ello se aplicarán diversas funciones sobre la misma para obtener la serie continua deseada.

Una vez aplicado los filtros necesarios obtenemos la siguiente serie de la Fig. 27.

En la Fig. 28 si bien siguen apareciendo ceros, aparecen en menor cantidad siendo originados en horarios de producción mínima como el horario de inicio y finalización.

### Serie temporal multivariada

El mismo análisis realizado para la serie univariada debe extrapolarse a la cantidad de dimensiones que consideraremos necesarias.

En la fig. 29 observamos la cantidad de defectos que luego son desglosados por departamento, lo que nos daría 8 *features*.

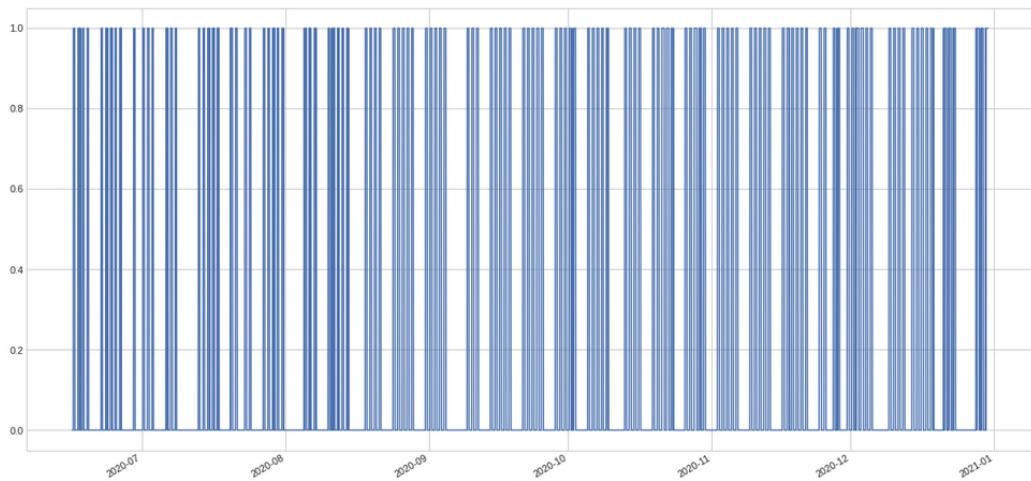


Figura 26: Gráfico temporal de detección de intermitencia.

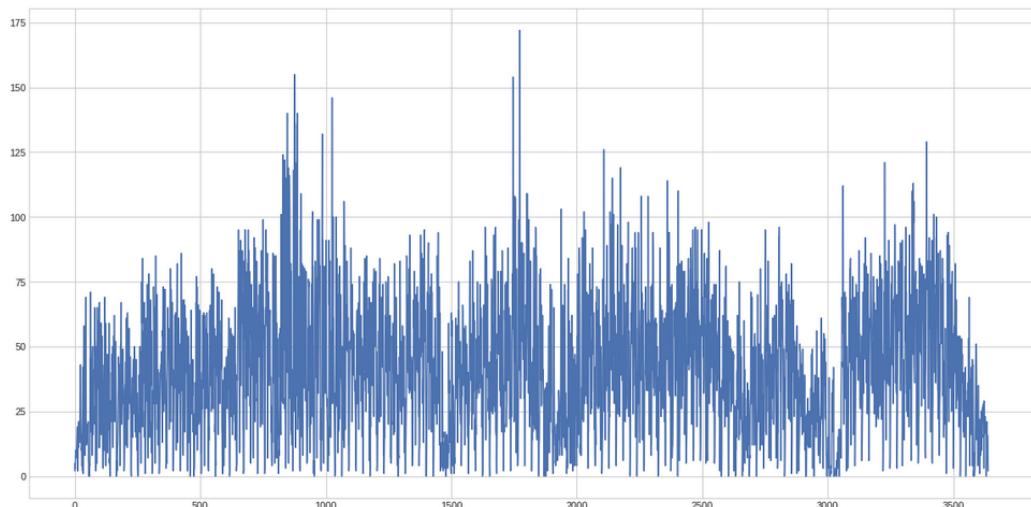


Figura 27: Gráfico temporal de cantidad de defectos procesado.

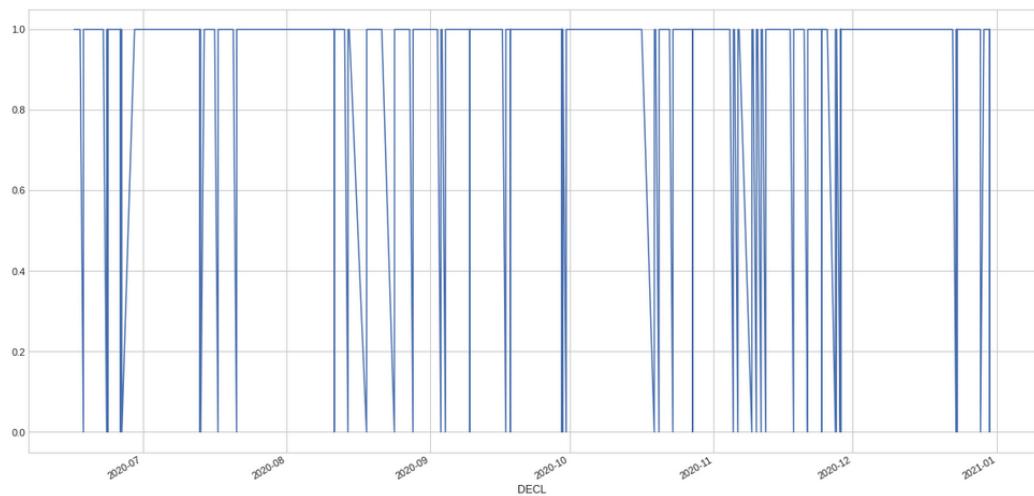


Figura 28: Gráfico temporal de detección de intermitencia procesado.

En la fig. 30 los defectos son desglosados por tipo, lo que nos daría 11 *features*. Lo que resultaría en 19 *features* que van a alimentar nuestro modelo.

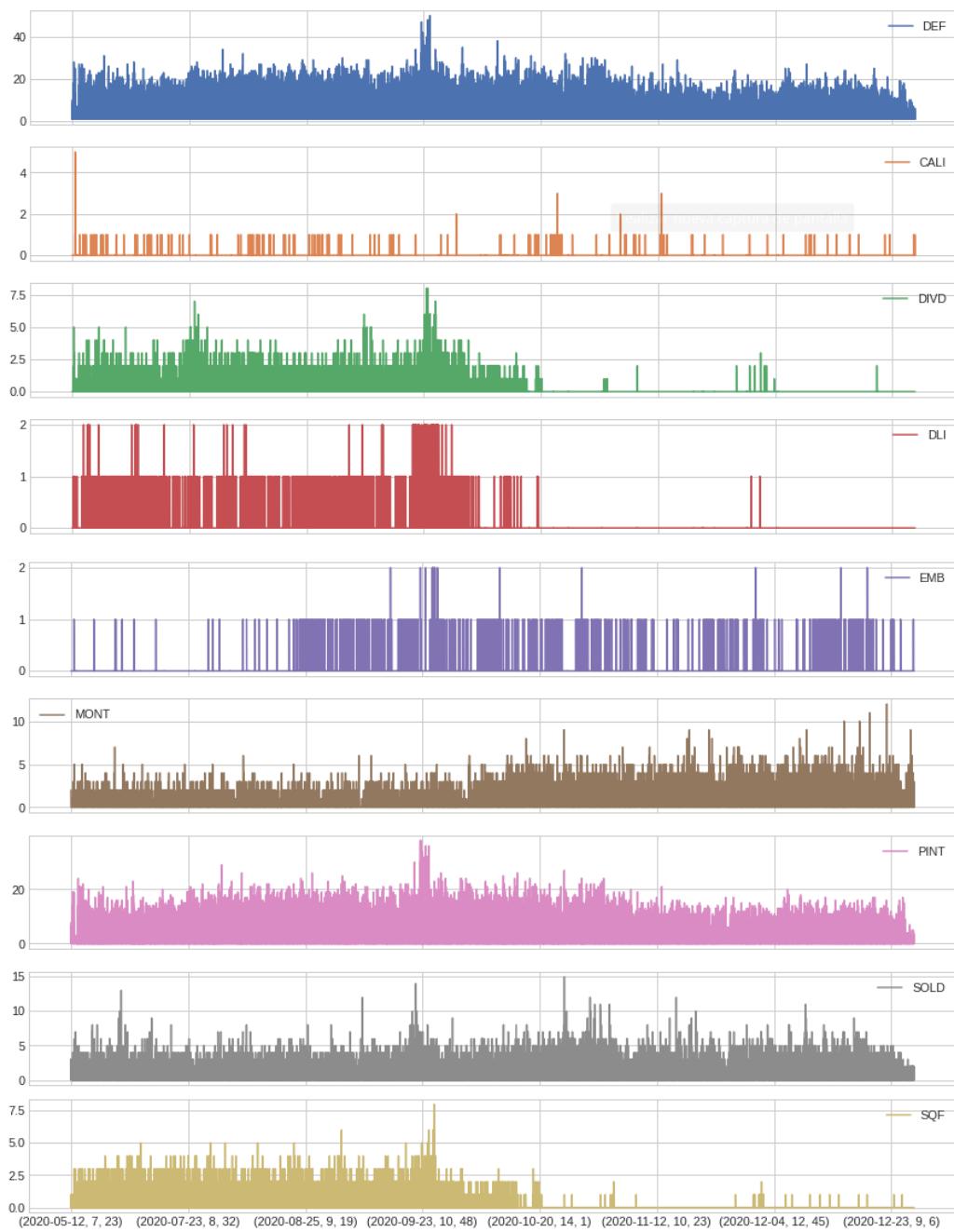


Figura 29: Gráfico temporal por departamentos.

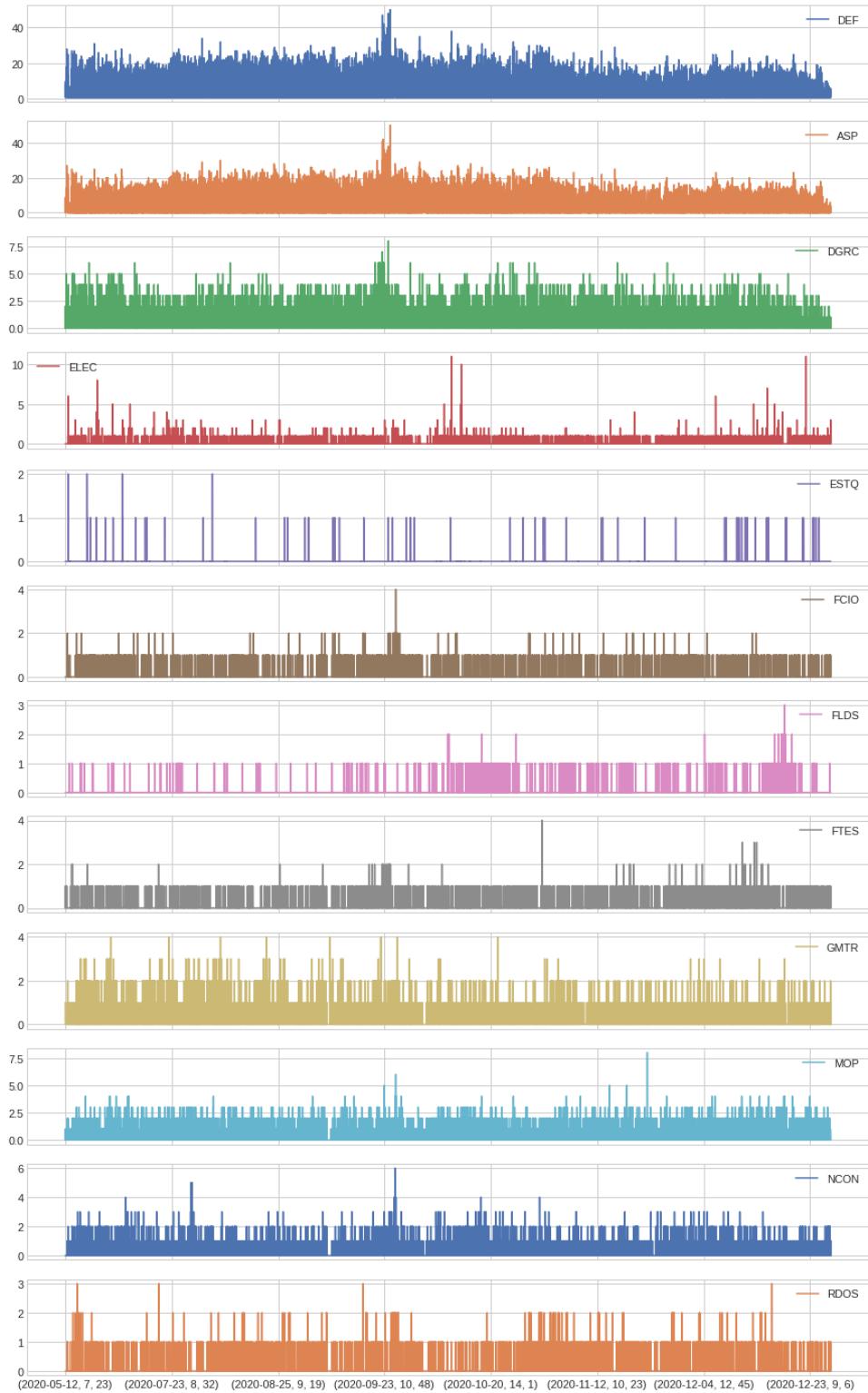


Figura 30: Gráfico temporal por tipo de defecto.

Tal como se realizó en la serie temporal univariada, se aplican los mismos pasos a estas series temporales de lo cual resultan las figuras 31 y 32.

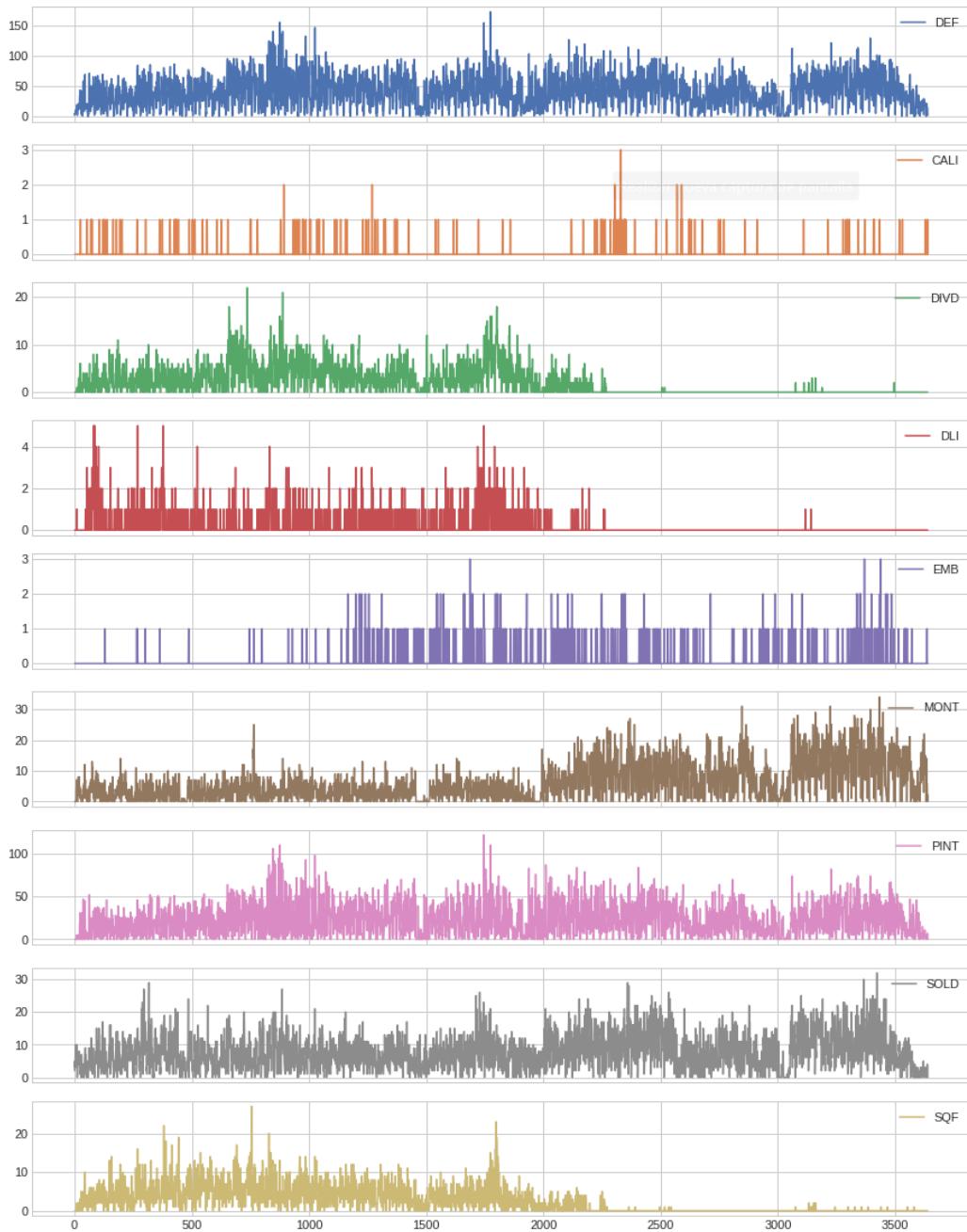


Figura 31: Gráfico temporal procesado por departamentos.



Figura 32: Gráfico temporal procesado por tipo de defecto.

Ya tenemos los *datasets* definitivos formateados de una manera correcta para alimentar nuestros modelos.

### 3.2.7 *Pipeline* de datos

Detengámonos a pensar sobre el tipo de problema con el que estamos tratando en el eventual caso que contáramos con los recursos para crear la arquitectura necesaria. En ingeniería de datos la arquitectura que orienta el flujo de datos se conoce como *pipeline* de datos.

El término *pipeline* de datos puede describir cualquier conjunto de procesos que mueven datos de un sistema a otro, a veces transformando los datos, a veces no. Básicamente, se trata de una serie de pasos en los que se mueven los datos. Este proceso puede incluir medidas como duplicación de datos, filtrado, migración a la nube y procesos de enriquecimiento de datos. [48]

#### Componentes

Las *pipelines* de datos contienen varios componentes, cada uno con un propósito específico, que facilitan el movimiento de datos:

- **Origen:** el origen representa la fuente de la que residen los datos originales.
- **Destino:** punto final al que se transfieren los datos, éste puede ser un almacén de datos (*Data Warehouse*), un punto final de API, una herramienta de análisis o más.
- **Flujo de datos:** movimiento de datos entre el origen y el destino.
- **Sistema de almacenamiento:** se refiere a todos los sistemas utilizados para preservar los datos a lo largo de las etapas del flujo de datos.
- **Procesamiento:** incluye todas las actividades involucradas en el movimiento de los datos.
- **Flujo de trabajo:** representa una serie de procesos junto con sus dependencias para mover datos a través del *pipeline*.
- **Monitoreo:** asegura que todas las etapas del *pipeline* estén funcionando correctamente.

#### *Pipeline ETL*

Los ETL son flujos de trabajo automatizados que llevan los datos del punto A al punto B y los transforman a lo largo del camino para mejorar su capacidad de análisis o que se parezcan más a lo que nosotros necesitamos. ETL se refiere a extraer (*Extract*), transformar (*Transform*) y cargar (*Load*) datos. [49]

En la fase de **extracción**, los datos a menudo se limitan a ser una instantánea de la base de datos en el momento actual o todos los datos históricos. Cuando extrae sus datos de la base de datos de una aplicación, a menudo los coloca en algún tipo de extracto CSV o JSON.

Estos extractos pueden provenir de tablas de bases de datos, APIs, registros, etc. Una vez que los datos se **transforman**, se **cargan** en el almacén de datos (*Data Warehouse*).

### **Pipelines de datos vs pipelines ETL**

La diferencia entre estos dos estructuras de flujo de datos radica principalmente en que un *pipeline* de datos es un término general del cual los *pipelines ETL* son un subconjunto. [48]

Característica	<i>Pipeline de datos</i>	<i>Pipelines ETL</i>
Carga de datos	No ocurre siempre. Si ocurre puede desencadenar nuevos procesos y flujos en otros sistemas.	Siempre termina en una base de datos o almacén de datos.
Transformación	No necesariamente.	Siempre.
Modo de trabajo	Tiempo real. Los datos están actualizándose constantemente.	En lotes. Usualmente se corre unas dos veces por día, generalmente cuando el tráfico es bajo.

Cuadro 10: Diferencias entre pipelines de datos y ETL.

### **DAG**

Usar la metáfora del *pipeline*, ayuda a explicar un proceso que de otro modo sería complejo. Pero los datos no están literalmente en un solo *pipeline*, comenzando por un lado y saliendo por el otro, pero esto nos sirve para destacar algunas características: [50]

- Cuando los datos se mueven a través de pipelines, se aíslan de otros datos.
- Los datos deben tratarse en varias etapas antes de que lleguen a su destino.

Pero esto lleva a una linealidad en el flujo de datos que podría hacer que una estructura no pueda ser dinámica. Una solución a esta problemática son los *DAGs* (*Directed Acyclic Graphs*), grafos acíclicos dirigidos.

Clarifiquemos el concepto a través de la fig. 33 que expone el caso de una multinacional que recibe pagos en diferentes países, dónde en cada uno los clientes pagan en su moneda local. Para crear los reportes es necesario unificar las ventas de cada país teniendo en cuenta las diversas divisas.

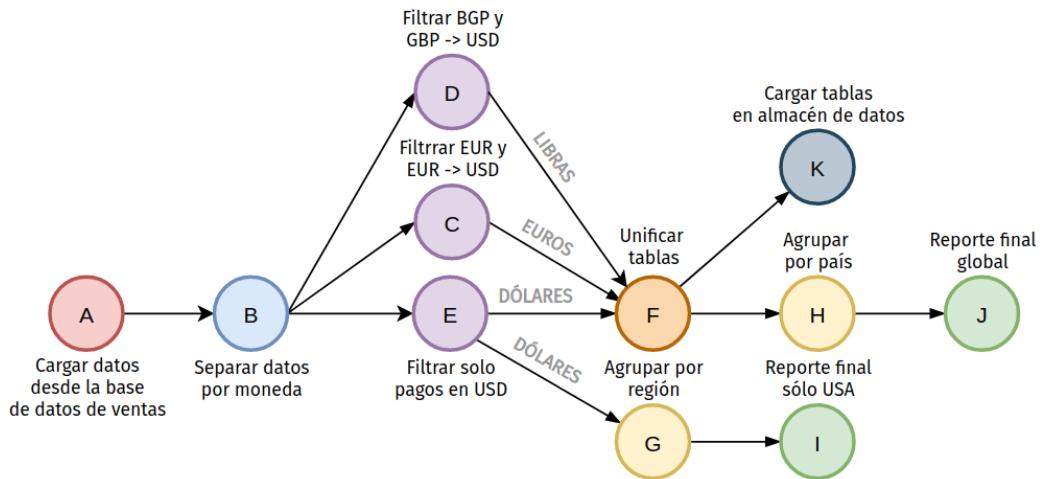


Figura 33: Ejemplo de grafo acíclico dirigido aplicado a caso real.

En matemáticas, un grafo es un conjunto finito de nodos, con vértices que conectan los nodos entre sí. En el contexto de la ingeniería de datos, cada nodo de un gráfico representa una tarea de procesamiento de datos y cada vértice (línea) tiene una dirección específica (indicada por la flecha) que conecta diferentes nodos. Ésta es la cualidad clave de un grafo dirigido: los datos solo pueden seguir la dirección del vértice. Además tener en cuenta que los nodos nunca se vuelven autorreferenciales, *i.e.* nunca pueden informarse a sí mismos, ya que esto podría crear un bucle infinito.

Resumiendo un *DAG* se caracteriza por ser:

- **Grafo:** todas las tareas se presentan en una estructura clara, con procesos discretos que ocurren en puntos establecidos y relaciones transparentes con otras tareas.
- **Acíclico:** ninguna tarea puede crear datos que se refieran a sí misma. Podría causar un bucle infinito y eso podría causar problemas.
- **Dirigido:** si existen varias tareas, cada una debe tener al menos una tarea definida en sentido ascendente o descendente, aunque podría tener ambas.

### Categorización del problema

Observemos la figura 34 para ponernos en contexto. Esta figura es un *DAG* del flujo de datos de nuestro sistema, dónde cada nodo es representado por las notebooks que se detallaron con anterioridad.

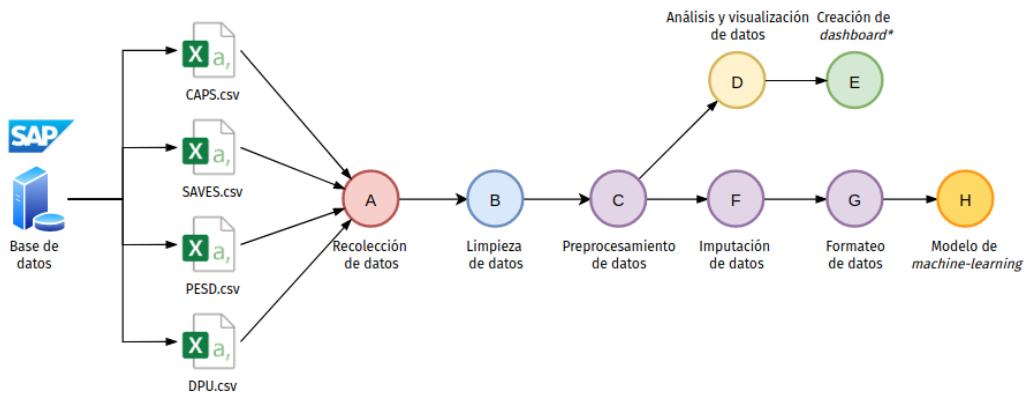


Figura 34: DAG del flujo de datos.

En primera instancia los datos son obtenidos (nodo A, fase de extracción) desde archivos .csv debido a que por políticas de la compañía, los datos se guardan en bases de datos de SAP y los empleados del departamento realizan extracciones diarias de forma manual para actualizar estos archivos. No es posible realizar consultas SQL o llamadas a una API (por nombrar algunos métodos de extracción de datos), por lo cuál este proceso no se puede automatizar.

Esto deviene en un impedimento crítico hacia nuestro sistema ya que por la naturaleza del problema a tratar (series temporales) es indispensable que el flujo de datos al modelo sea constante y en tiempo real (*streaming* de datos).

Los nodos B, C, F y G corresponderían a la etapa de transformación, no obstante estos datos no son almacenados por tanto nuestro flujo de datos no encuadra en un *pipeline* ETL.

Los nodos D y E\* (esta notebook no fue creada debido a que no se solicitó) corresponden a las etapas de análisis y visualización. Un punto a destacar es que esta etapa sería de gran valor para la compañía ya que permite visualizar datos de un modo que antes no era posible (recordar que los datos debieron pasar por los nodos B y C para llegar a esta instancia), lo que lleva a una reducción de costes debido a mayor reactividad en la detección de defectos.

En última instancia está el modelo, que recibe los datos en un formato acorde, se encarga de procesarlos y realiza la inferencia. Quitando los nodos D y E, nuestro problema se ajusta a la categoría de *pipeline* de datos.

## 4 Modelos

Los modelos serán creados combinando las diferentes capas que se han definido con anterioridad.

### 4.1 Métricas

Es importante definir métricas para determinar el rendimiento de nuestro modelo de una manera empírica, más allá de que nuestros modelos serán entrenados utilizando como métrica de pérdida MAE (error absoluto medio).

#### 4.1.1 MAPE

Una de las métricas más habituales para medir la precisión del pronóstico de un modelo es MAPE, que significa error porcentual absoluto medio (*mean absolute percentage error*). [51]

La fórmula para calcular MAPE es la siguiente:

$$\text{MAPE} = \frac{100}{N} \times \sum_{i=1}^N \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

dónde:

- $y_i$ : valores de los datos reales.
- $\hat{y}_i$ : valor de datos pronosticados.
- $N$ : cantidad de puntos.

MAPE se usa generalmente debido a su fácil interpretación y simple explicación. Por ejemplo, un valor MAPE de 11,5% significa que la diferencia promedio entre el valor pronosticado y el valor real es 11,5%. Cuanto menor sea el valor de MAPE, mejor podrá un modelo pronosticar valores.

Sin embargo esta métrica no nos será de utilidad ya que si alguno de los valores reales es 0 (como sucede) entonces el valor será incalculable por la división por 0.

#### 4.1.2 MASE

En el pronóstico de series temporales, el MASE es el error medio absoluto escalado (*Mean Absolute Scaled Error*). Es una métrica para determinar la efectividad de los pronósticos generados a través de un algoritmo al comparar las predicciones con el resultado de un enfoque de pronóstico ingenuo o *naive forecast*. [52]

El *naive forecast* se genera en cualquier paso equiparando el pronóstico actual con la salida del último paso de tiempo. Por ejemplo, la predicción de las ventas de una empresa al comienzo de un mes se realiza comparándola con las ventas reales del último mes sin considerar ningún patrón estacional.

El MAE para *naive forecast* se calcula de la siguiente forma:

$$\text{MAE}_{\text{naive}} = \frac{1}{N-1} \sum_{i=2}^N |y_i - y_{i-1}|$$

Para determinar la efectividad de un algoritmo de pronóstico, el MASE se calcula de la siguiente manera:

1. Calcular el MAE (*Mean Absolute Error*) para los pronósticos del algoritmo.

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^N |\hat{y}_i - y_i|$$

2. MASE está dado por el ratio entre el MAE del algoritmo y el MAE del *naive forecast*.

$$\text{MASE} = \frac{\text{MAE}}{\text{MAE}_{\text{naive}}}$$

## Características

- MASE da una indicación de la eficacia del algoritmo de pronóstico con respecto a un *naive forecast*. Un valor mayor a uno indica que el algoritmo está funcionando mal en comparación al *naive forecast*.
- MASE es inmune al problema que enfrenta MAPE cuando la salida de la serie de tiempo real en cualquier paso de tiempo es cero. Sin embargo, se observa que para una serie de tiempo con todos los valores iguales a cero en todos los pasos, la salida de MASE tampoco se definirá, pero tales series de tiempo no son realistas.
- MASE es independiente de la escala del pronóstico, ya que se define utilizando la proporción de errores. Esto significa que los valores de MASE serán similares si pronosticamos series de tiempo de alto valor, como el número de paquetes de tráfico de Internet que cruzan un router por hora, en comparación con el número de peatones que cruzan un semáforo ocupado cada hora.

## 4.2 Framework

Al momento de definir el *framework* de *Machine Learning* podemos encontrar 2 alternativas que se disputan el mayor porcentaje del mercado:

- *TensorFlow*: es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por *Google* para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. Usualmente se utiliza como *back-end* de otro *framework* llamado *Keras* que posee una API más amigable con el usuario.
- *PyTorch*: es una biblioteca de aprendizaje automático de código abierto basada en la biblioteca de *Torch*, utilizado para aplicaciones del campo de la visión artificial y del procesamiento natural del lenguaje, principalmente desarrollado por el Laboratorio de Investigación de Inteligencia Artificial de *Facebook*.

Los datos recopilados por la página *PapersWithCode* [53] en la fig. 35 nos muestra una clara tendencia de los investigadores a preferir *PyTorch* sobre los demás *frameworks*.

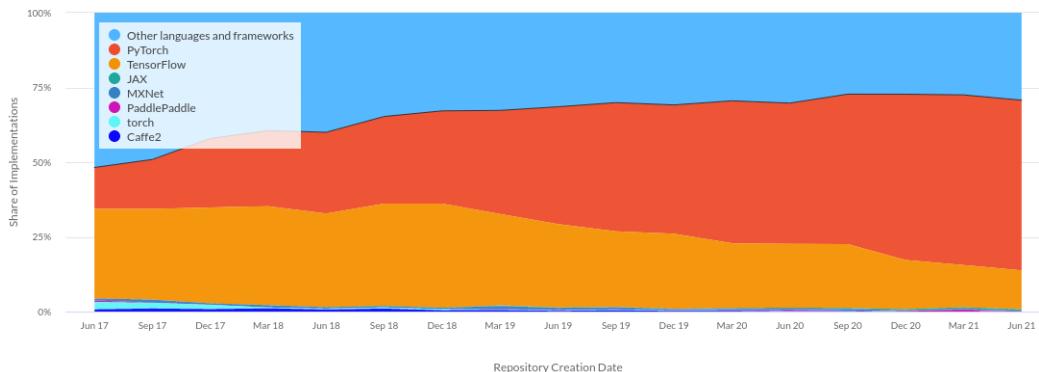


Figura 35: Porcentaje de implementaciones de cada *framework* en *papers*.

A pesar de esto para nuestra tarea debido a su facilidad de uso se escogió *Keras*, biblioteca de redes neuronales de código abierto cuyo autor principal es el ingeniero François Chollet.

Está especialmente diseñada para posibilitar la experimentación en un corto plazo con redes de Aprendizaje Profundo. Sus fuertes se centran en ser amigable para el usuario, modular y extensible. [54]

#### 4.2.1 Callbacks

Una característica muy interesante que nos ofrece *Keras* son las *callbacks*. Se define como un objeto que puede realizar acciones en varias etapas del entrenamiento (por ejemplo, al comienzo o al final de una época, antes o después de un solo lote, etc.). Repasaremos algunas de las que fueron utilizadas en el entrenamiento del modelo. [55]

- **EarlyStopping:** detiene el entrenamiento cuando una métrica monitoreada haya dejado de mejorar. Asumimos que el objetivo de un entrenamiento es minimizar la pérdida, *e.g.* la métrica a monitorear sería `loss` y el modo sería `min`. Un ciclo de entrenamiento verificará al final de cada época si la pérdida ya no está disminuyendo, considerando el `min_delta` y la paciencia (cantidad de épocas sin mejorar). Una vez que la métrica no disminuye, el entrenamiento termina. Esto es ventajoso para ahorrar tiempo de entrenamiento.
- **ReduceLROnPlateau :** reduce la tasa de aprendizaje cuando una métrica ha dejado de mejorar. Los modelos usualmente se benefician de la reducción de la tasa de aprendizaje en un factor de 2 a 10 una vez que el aprendizaje se estanca. Esta *callback* monitorea una métrica y si no se ve ninguna mejora en un número de épocas de "pacienza", la tasa de aprendizaje se reduce. Ha brindado excelentes resultados.
- **ModelCheckpoint:** trabaja en conjunto con el entrenamiento para guardar un modelo o pesos (en un archivo de punto de control) en algún intervalo, por lo que el modelo o los pesos se pueden cargar más tarde para continuar el entrenamiento desde el estado guardado. Algunas opciones que ofrece:
  - Mantener solo el modelo que ha logrado el "mejor desempeño" hasta ahora, o guardar el modelo al final de cada época sin importar el desempeño.
  - La frecuencia a la que debería guardar. Actualmente, se permite guardar al final de cada época o después de un número fijo de lotes de entrenamiento.
  - Si solo se guardan los pesos o se guarda todo el modelo.

### 4.3 Arquitecturas

Se define como la arquitectura del modelo a la disposición del mismo en lo que concierne a sus capas y al flujo de la información a través de las mismas.

En *Keras* es posible definir un modelo **Sequential**, lo que significa que cada salida alimentará a la siguiente y no habrá bifurcaciones entre el flujo de los tensores. También se podría resumir que **Sequential** agrupa una pila lineal de capas, lo que para nuestro propósito es suficiente. Esto se denomina arquitectura de tipo lineal. A modo informativo en *TensorFlow* es posible crear arquitecturas no-lineales que admitan múltiples flujos de datos entre las diferentes capas.

#### 4.3.1 Hiperáparametros de capas

En la figura 36 podemos observar las capas disponibles para crear nuestra arquitectura.

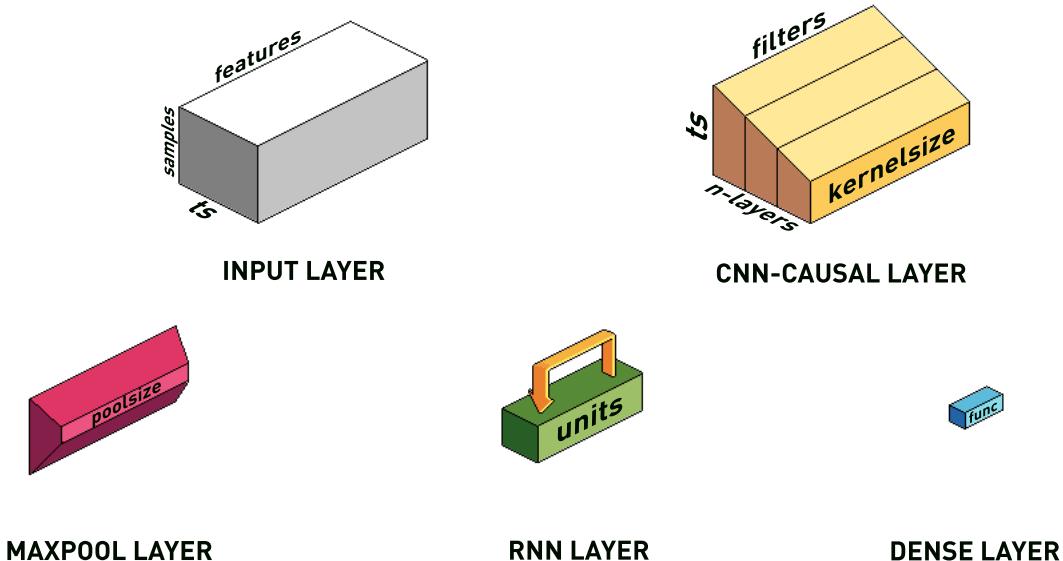


Figura 36: Tipos de capas disponibles.

Detallaremos los hiperáparametros que será posible *tunear* de cada capa.

- **INPUT:**
  - **ts**: *timestamps*.
  - **features**: características.
  - **samples**: cantidad de muestras.

- CNN-CAUSAL
  - **n-layers**: cantidad de capas, podemos escalar en la profundidad de CNNs.
  - **filters**: filtros.
  - **kernelsize**: tamaño del *kernel*.
  - **ts = timestamps** o rodajas de tiempo.
- GRU y LSTM
  - **units**: unidades de la celda.
- MAXPOOL:
  - **poolszie**: tamaño de *pool*.
- FC
  - **func**: función de activación.

#### 4.3.2 Tipos

En función a las capas explicadas en el marco teórico se ensayarán 5 arquitecturas de modelo representadas de forma visual en la fig. 37, con el objetivo de identificar la de mejor desempeño utilizando como parámetro la métrica MASE.

- CNN: CNN → FC
- LSTM: LSTM → FC
- GRU: GRU → FC
- CNNLSTM: CNN → MAXPOOL → LSTM → FC
- CNNGRU: CNN → MAXPOOL → GRU → FC

En el caso de las últimas dos redes se utilizó la capa de MAXPOOL para evitar el cuello de la botella de los datos, corroborando que no se afectó el rendimiento del modelo y mejoraron los tiempos de entrenamiento.

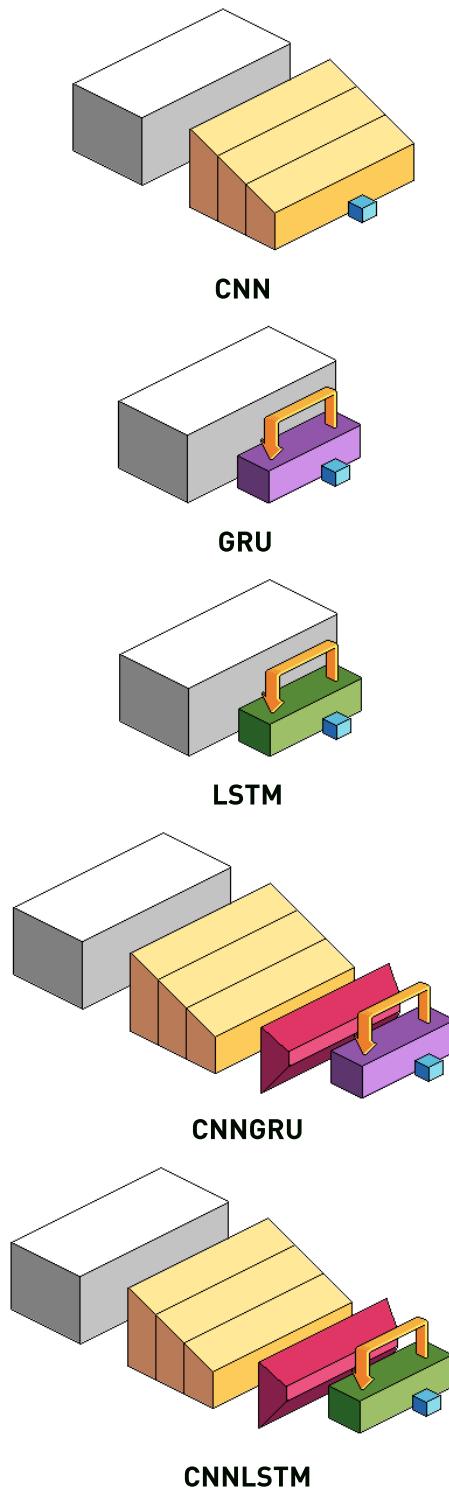


Figura 37: Tipos de arquitecturas.

## 4.4 Entrenamiento

### 4.4.1 Datos de prueba

Ya tenemos todos los ingredientes necesarios para empezar el entrenamiento de nuestra red neuronal, sin embargo sería coherente utilizar datos de prueba o *dummy data* para corroborar que todo funcione según lo esperado.

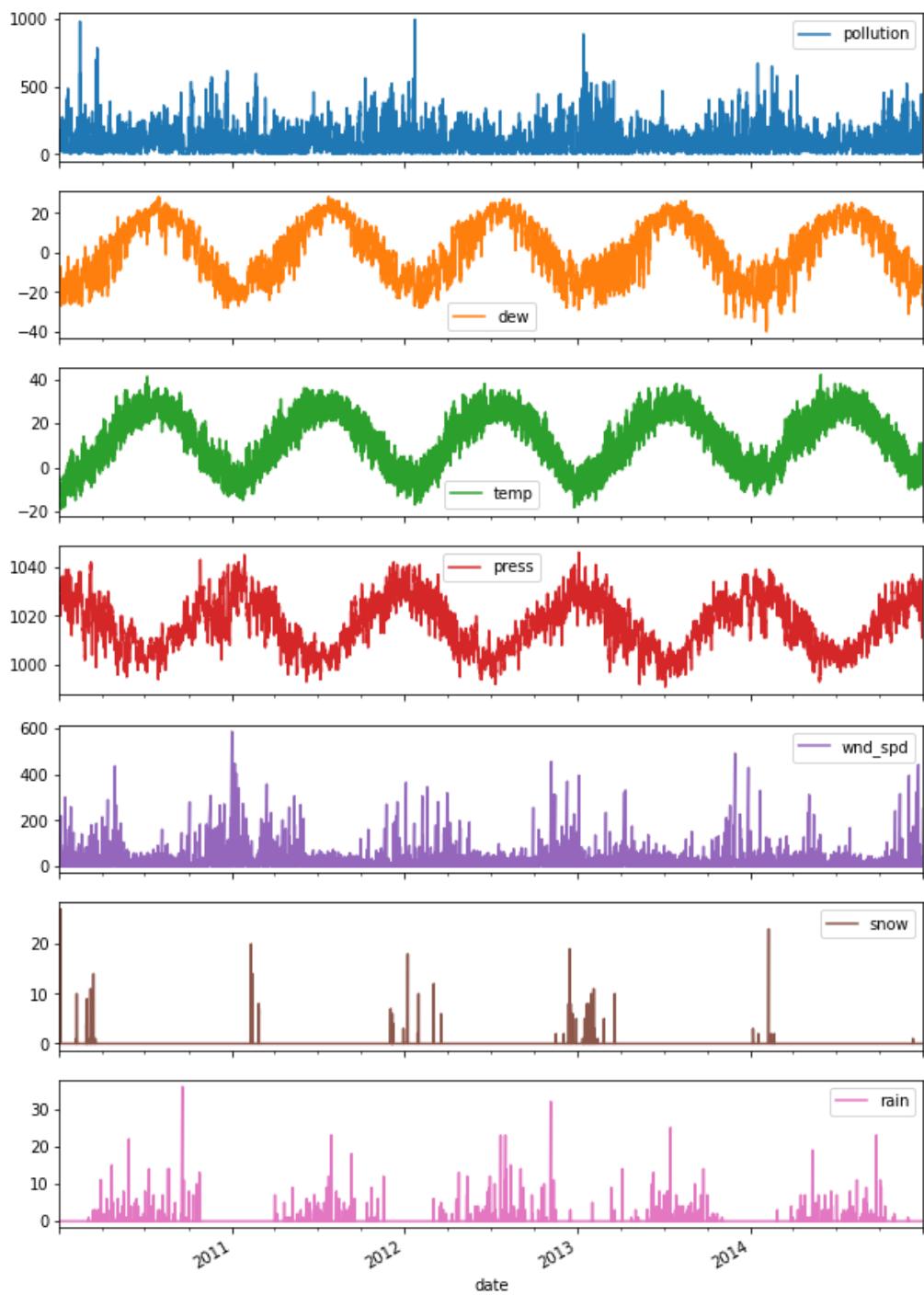
Para esta tarea haremos uso de un *dataset* de los niveles de polución en el aire. Contiene detalles sobre diferentes parámetros que son monitoreados cada hora durante cinco años en la embajada de Estados Unidos en Beijing, China.

Los datos incluyen la fecha y hora, la contaminación denominada concentración de PM2.5 y la información meteorológica. La lista completa de características en los datos es la siguiente:

- **pollution**: concentración de PM2.5, son partículas muy pequeñas suspendidas en el aire que tienen un diámetro de menos de 2.5 micras. Será nuestro *target*.
- **dew**: punto de rocío, es la más alta temperatura a la que empieza a condensarse el vapor de agua contenido en el aire, produciendo rocío, neblina, cualquier tipo de nube o, en caso de que la temperatura sea lo suficientemente baja, escarcha.
- **temp**: temperatura.
- **press**: presión.
- **wnd\_dir**: dirección del viento combinada.
- **wnd\_spd**: velocidad del viento acumulada.
- **snow**: horas acumuladas de nieve.
- **rain**: horas acumuladas de lluvia.

Bajo estos datos se enmarcará un problema de pronóstico en el que dadas las condiciones climáticas y la contaminación de las horas previas, pronosticamos la contaminación de la hora siguiente.

En la fig. 38 podemos observar que nuestros datos son bastante armoniosos notando 3 ondas sinusoidales, característica que el modelo explotará (pero la mayoría de las series temporales en la vida real no serán tan convenientes).

Figura 38: *Dataset* de contaminación del aire.

Luego deberemos dividir nuestro *dataset* como vimos con anterioridad en 3 sets en la siguiente proporción:

- **train** (*entrenamiento*): 70%.
- **validation** (*validación*): 20%.
- **test** (*prueba*): 10%.

Recordar que la métrica `val_loss` será aplicada sobre el set de validación y MASE sobre el set de prueba.

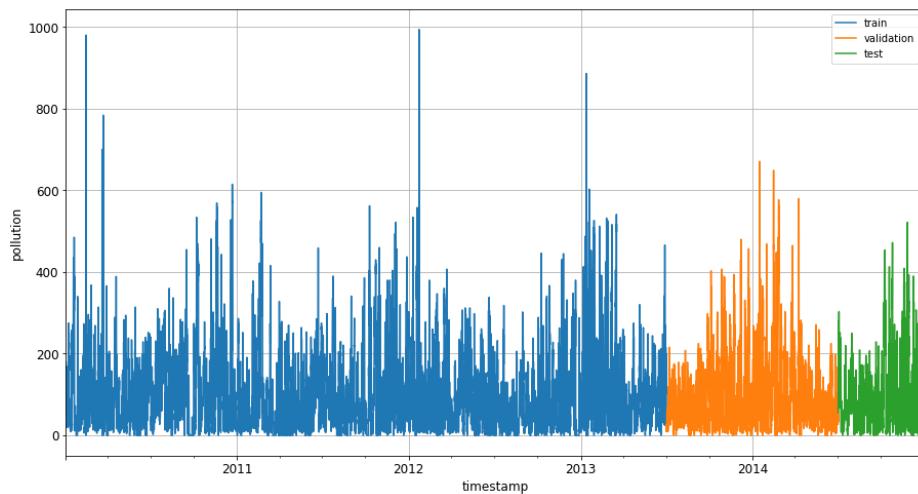


Figura 39: *Dataset* de contaminación del aire.

#### 4.4.2 Desplazamiento temporal

Una operación importante a realizar en nuestro dataset es el desplazamiento temporal o *time-shifting* para lograr que nuestra red pueda aprender sobre los *timesteps* pasados de nuestras features y comparar su salida contra los *timesteps* futuros en caso de ser la *target feature*.

Para aclarar esta idea pongamos nuestro enfoque en la figura 40. Definiremos  $T$  a la cantidad de *timesteps* que queremos que nuestro modelo utilice para extraer información pasada, además del actual.

Y definiremos **HORIZON** como al horizonte de pronóstico, *i.e.* la cantidad de *timesteps* futuros que queremos predecir de nuestra característica objetivo o *target feature*.

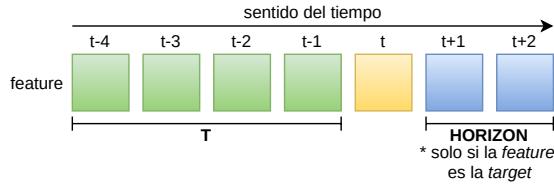


Figura 40: Desplazamiento temporal visualizado.

#### 4.4.3 Tipo de horizonte

Un punto importante a aclarar es que es diferente la arquitectura a implementar según si nuestra *target feature* es de horizonte único o multi-horizonte.

##### Horizonte único

Veremos en una perspectiva más amplia el funcionamiento de nuestro modelo para tomar noción de donde nos encontramos y para esta tarea nos será útil la fig. 41 [56]. Básicamente tomamos los  $T$  *timesteps* de nuestras *features* para alimentar a la arquitectura (que fueron presentadas en la sección 4.3), en este caso compuesta de celdas RNN, que luego la salida alimenta una neurona que determina el valor de  $t+1$ .

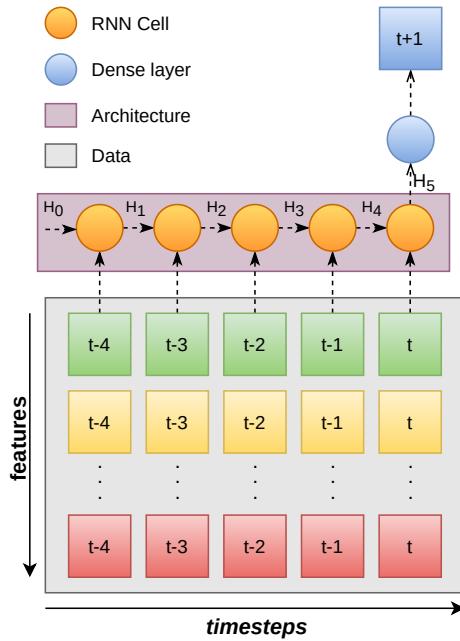


Figura 41: Modelo de horizonte único.

### Horizonte múltiple

Observemos la fig. 42 [57]. Si determinamos que nuestra *target feature* será de horizonte múltiple, debemos aumentar la complejidad de nuestra arquitectura lo que se traduce en menor personalización en lo que se refiere a adición de capas.

Debemos contar con dos capas **encoder** y **decoder**, donde una se encargará de procesar los datos de entrada y la otra de decodificar la salida de la primera para obtener la forma de tensor deseada a la salida de la red, lo que sería  $t+1$ ,  $t+2$  y  $t+3$  en nuestro ejemplo.

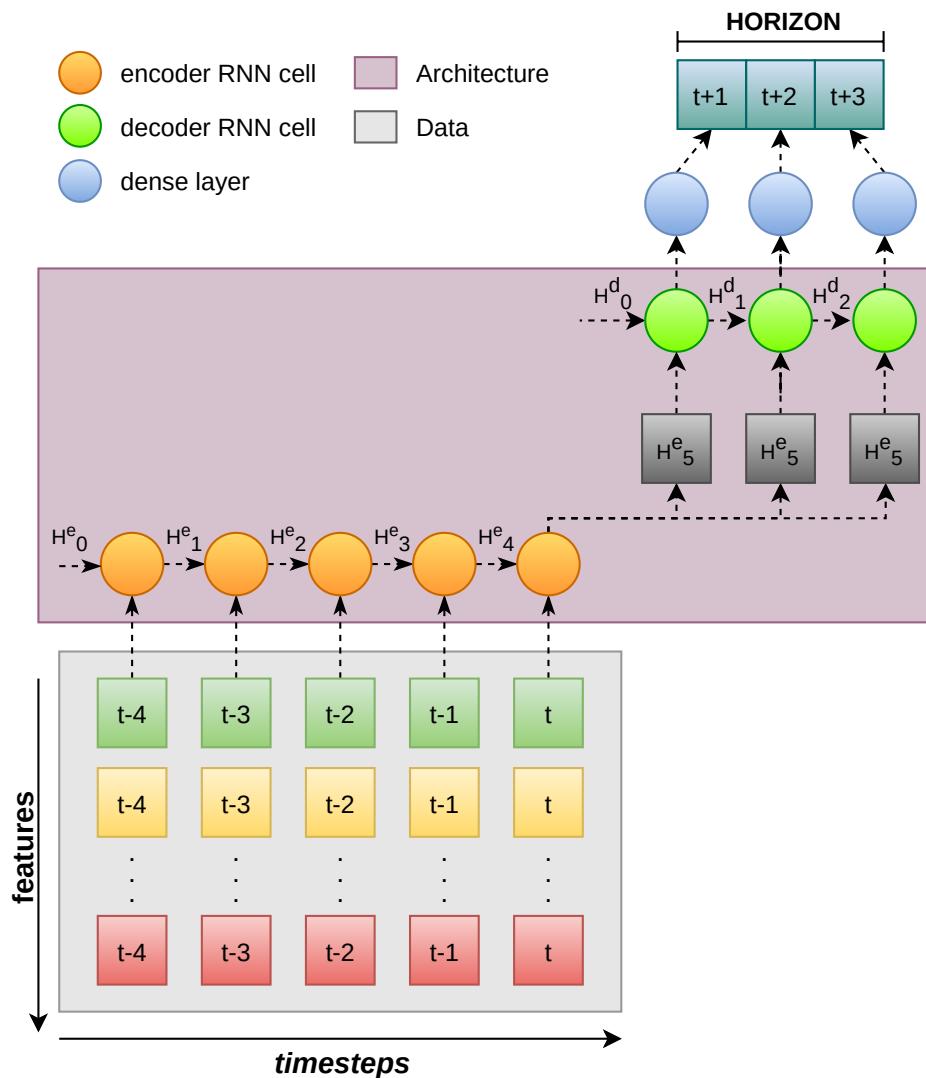


Figura 42: Modelo de horizonte múltiple.

En la fig. 43 podemos observar la salida de un pronóstico de horizonte múltiple con  $T=16$  y  $HORIZON=4$ . Notamos que a medida que queremos predecir pasos más lejanos perdemos precisión, en la tabla 11 se corrobora de manera numérica esta observación.

Además cuando veamos los resultados de los modelos de horizonte único notaremos que  $t+1$  tiene mayor precisión que en este tipo de modelos.

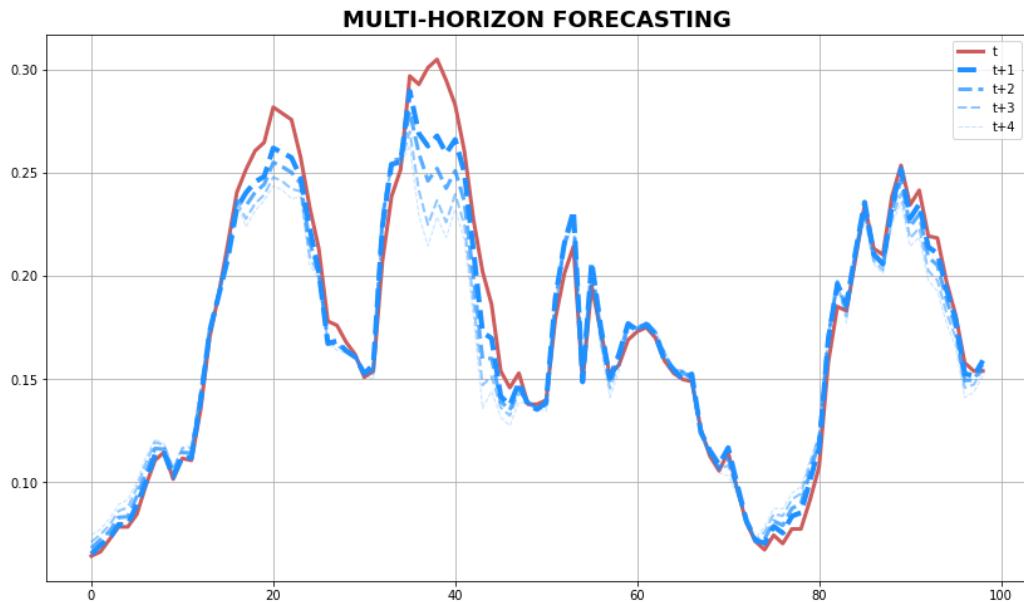


Figura 43: Pronóstico de horizonte múltiple.

$t$	MAE	MASE
$t+1$	0.00527	0.454
$t+2$	0.00728	0.628
$t+3$	0.00953	0.822
$t+4$	0.01167	1.006

Cuadro 11: Resultados del modelo de horizonte múltiple.

#### 4.4.4 Resultados

Se entrenaron los 4 modelos principales (posteriormente se mostrarán los resultados de la CNN causal dilatada, ya que requiere un tratamiento diferente). Los siguientes hiper-parámetros se seleccionaron luego de varios

ensayos donde se comprobó que como valores estándar tenían un rendimiento aceptable:

- EPOCHS=200
- UNITS=250
- BATCH\_SIZE=64
- T=8
- HORIZON=1
- CNN\_LAYERS=3

Se obtuvieron los resultados de la fig. 44. Los saltos que observamos en el `val_loss` (el más notorio en CNNLSTM) se deben a la callback de `ReduceLROnPlateau` (sección 4.2.1), siendo una función de *Keras* muy provechosa para nuestro objetivo.

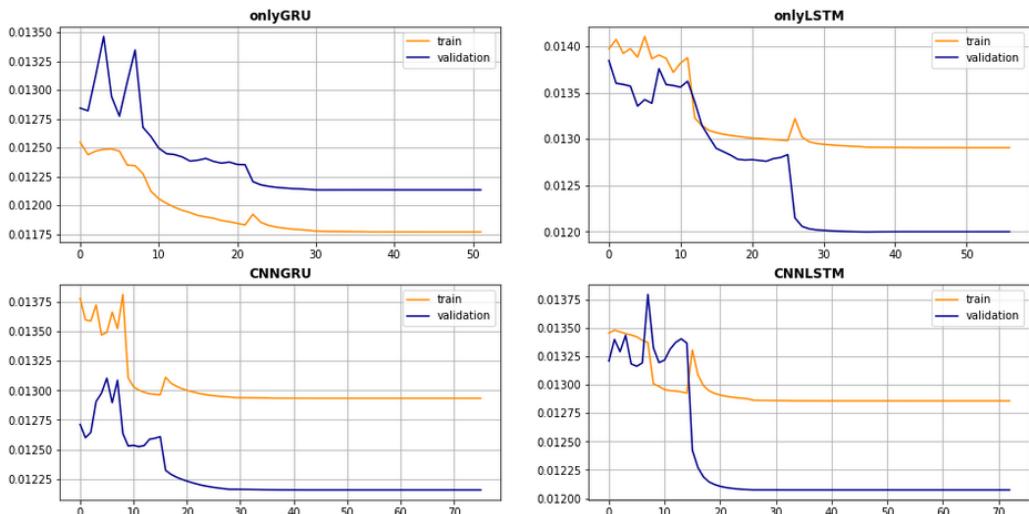


Figura 44: Resultados de entrenamiento de los modelos GRU/LSTM.

Para el entrenamiento de modelos compuestos exclusivamente de capas CNN dilatadas causales se utilizaron los siguientes hiper-parámetros:

- KERNEL\_SIZE=2
- DEEP\_LAYERS=1 +  $i$  donde  $i \in \mathbb{N} : 1 \leq i \leq 8$ .
- `dilation_rate`= $i^2$  donde  $i \in \mathbb{N} : 1 \leq i \leq 8$ .

En la fig. 45 representamos cada uno de los entrenamientos de los modelos, en los cuáles fuimos variando en cada modelo el valor de `dilation_rate`.

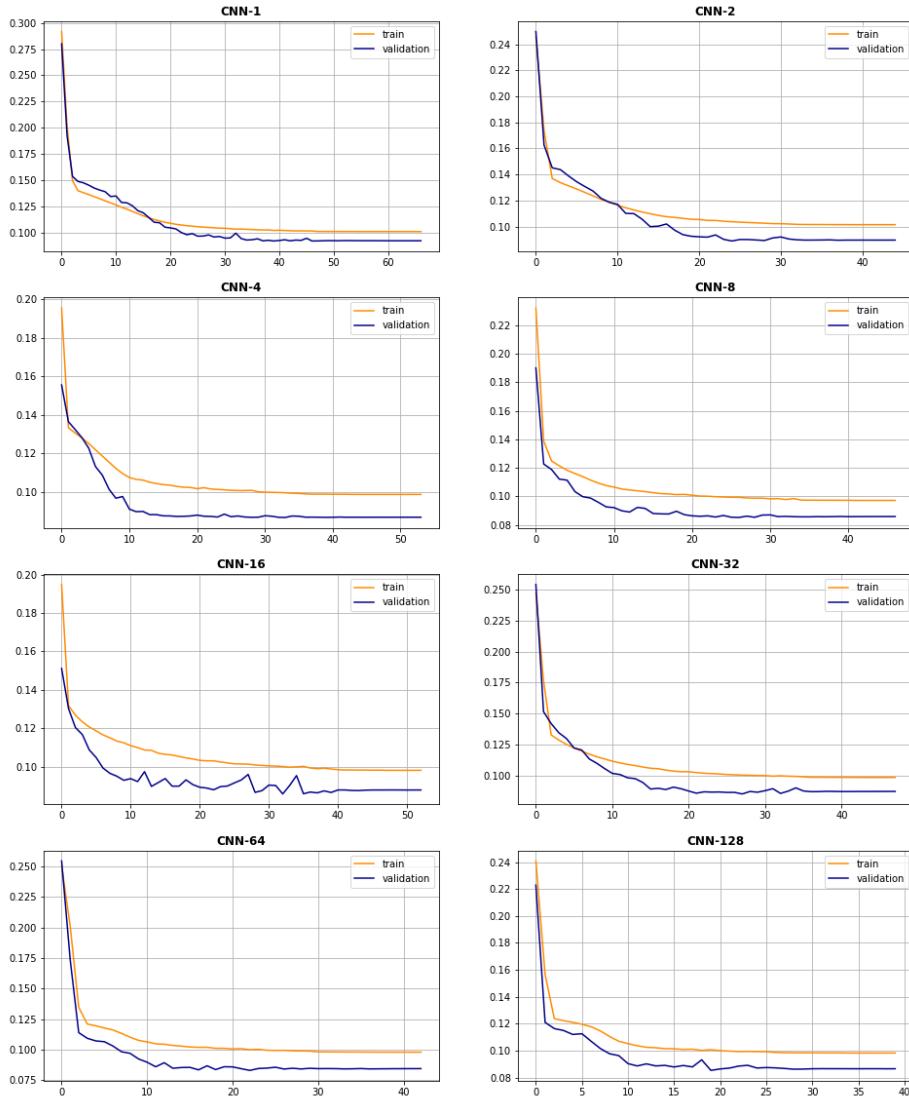


Figura 45: Resultados de entrenamiento de los modelos CNN.

Las diferencias que podemos notar entre la fig. 44 y la fig. 45 es que los modelos que poseen un componente RNN tienden a ser mucho más caóticos en su entrenamiento pero si la cantidad de épocas es suficiente el resultado es satisfactorio. En cambio, los modelos CNNs puros reflejan un entrenamiento más armonioso y continuo.

En la tabla 12 y en la fig. 46 verificamos que el modelo con el mejor MASE es el CNNGRU, sin embargo esto lo realizamos con valores fijos y estándares para los hiper-parámetros de los modelos.

model	MAE	MASE
GRU	0.0121	0.2894
LSTM	0.0123	0.2777
CNNGRU	<b>0.0123</b>	<b>0.2775</b>
CNNLSTM	0.0123	0.2793
CNN-1	0.0144	0.6526
CNN-2	0.0124	0.3476
CNN-4	0.0126	0.3242
CNN-8	0.0133	0.4389
CNN-16	0.0124	0.3172
CNN-32	0.0127	0.3318
CNN-64	0.0126	0.3597
CNN-128	0.0122	0.3121

Cuadro 12: Métricas de los modelos.

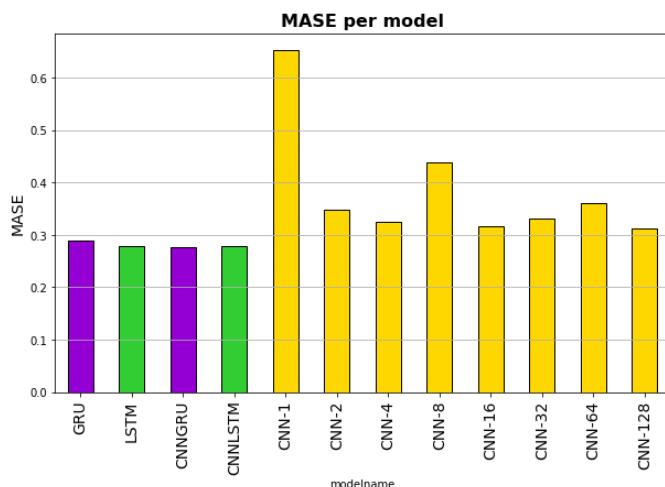


Figura 46: MASE por modelo.

Para una mejor visualización de los resultados, en la fig. 47 se toman las 200 primeras muestras. La precisión del modelo es alta, pronosticando valores muy cercanos a los reales. Por tanto, para estos datos el modelo ha resultado eficaz.

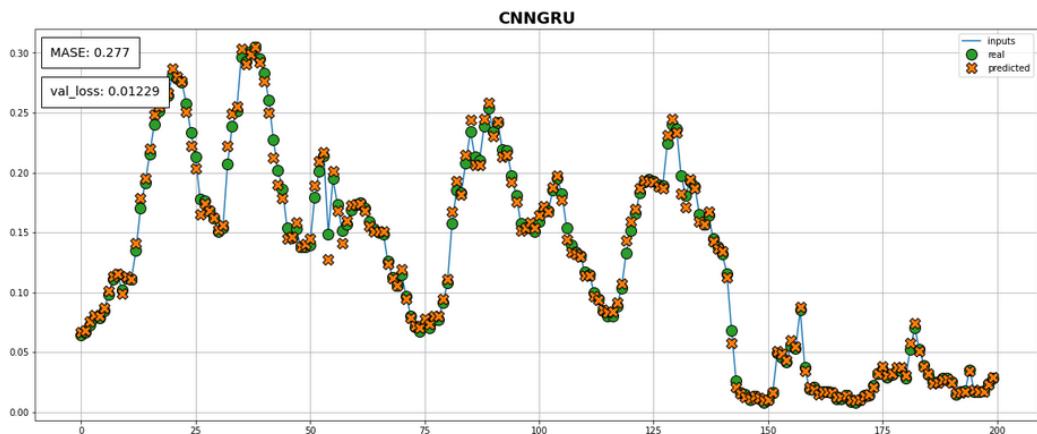


Figura 47: Pronóstico del mejor modelo.

## 4.5 Ajuste de hiperparámetros

Para obtener el mejor modelo es necesario explorar diversas configuraciones a través del ajuste de hiperparámetros (sección ??).

Para esta labor existen diversas librerías detalladas en la tabla 13 [58]. Entre todas ellos nuestra decisión será *Keras-tuner*, esencialmente por su facilidad de uso con la librería *Keras* (*framework* seleccionada de *Machine Learning*).

<b>Frameworks HPO</b>	<b>Algoritmos</b>			<b>Frameworks ML</b>						<b>GPU</b>
	HB	BO	RS	PT	TF	SL	XGB	LGBM	MN	
<i>Ray Tune</i>	Si	Si	No	Si	Si	Si	Si	Si	No	Si
<i>Optuna</i>	Si	Si	No	Si	Si	No	No	No	Si	No
<i>Hyperopt</i>	No	No	Si	Si	Si	Si	Si	No	No	Si
<i>Scikit-Optimize</i>	Si	Si	Si	No	No	Si	No	No	No	No
<i>Keras-tuner</i>	Si	Si	Si	No	Si	No	No	No	No	Si
<i>Microsofts NNI</i>	No	Si	No	Si	Si	Si	No	No	No	Si

Cuadro 13: Comparación de los diferentes *frameworks* disponibles para optimización de hiper-parámetros.

Aconimos utilizados:

- HB: *HyperBand*.
- BO: *Bayesian Optimization*.

- RS: *Random Search*.
- PT: *PyTorch*.
- TF: *TensorFlow*.
- SL: *Scikit-Learn*.
- XGB: *XgBoost*.
- LGBM: *LightGBM*.
- MN: *MxNet*.

#### 4.5.1 *Keras-tuner*

La librería se compone de diversas clases y métodos que nos permitirán ajustar los hiperparámetros de nuestros modelos creados en *Keras*. [59]

##### **Clase HyperParameters**

Sirve como contenedor de hiperparámetros, una instancia de `HyperParameters` contiene información sobre el espacio de búsqueda y los valores actuales de cada hiperparámetro.

Además nos provee diversos métodos para decidir de que manera queremos explorar el espacio de búsqueda de un determinado hiperparámetro:

- `Boolean`: elige entre `True` o `False`.
- `Choice`: elige entre una lista de opciones.
- `Fixed`: fijo, no permite cambiar su valor.
- `Int`: explora números enteros entre el valor inicial y final, es posible seleccionar el modo y el paso.
- `Float`: *ídem* al ítem anterior, pero con números reales.

##### **Clase Oracle**

Cada clase `Oracle` (óraculo) implementa un algoritmo de ajuste de hiperparámetros particular. Un `Oracle` se pasa como argumento a un `Tuner` (clase cuyo objetivo es realizar la búsqueda de hiperparámetros). `Oracle` le informa al `Tuner` qué hiperparámetros deben probarse a continuación.

La mayoría de las clases de `Oracle` se pueden combinar con cualquier subclase de `Tuner` definida por el usuario. Si no se necesita una subclase de `Tuner` (el caso más común), también se proporciona una serie de clases que empaquetan un `Tuner` y un `Oracle` juntos (cómo veremos a continuación).

##### **Clase Tuner**

Su propósito es realizar la búsqueda de hiperparámetros en el espacio de búsqueda definido, y como se aclaró en la sección anterior es posible crearlos de forma personalizada pero los algoritmos ofrecidos son más que suficientes para nuestra tarea (analizados en la sección ??):

- Búsqueda Aleatoria (descartado).
- Optimización Bayesiana: parámetro a destacar:

- `max_trials`: número total de pruebas (configuraciones de modelo) para probar como máximo.
- *Hyperband*: parámetros a destacar:
  - `factor`: factor de reducción ( $\eta$ ) para el número de épocas y el número de modelos para cada `bracket`.
  - `hyperband_iterations`: el número de veces que se iterará sobre el algoritmo *Hyperband* completo. Una iteración ejecutará aproximadamente  $s_{max}(\log_\eta s_{max})^2$  épocas acumulativas en todas las pruebas (notación tomada de la sección ??). Se recomienda establecer esto en un valor tan alto como esté dentro de su presupuesto de recursos.

#### 4.5.2 Espacio de búsqueda

Para cada tipo de modelo se definió un espacio de búsqueda determinado en función a su arquitectura.

##### RNN

Se limitó a explorar la cantidad de unidades en las capas recurrentes e intercambiar la función de activación de la capa final.

```

1   # RNN layer
2   for i in range(8, 256, 16):
3       units = i
4       # Dense layer
5       for j in ['relu', 'sig', 'tanh']:
6           dense = j
7

```

##### CNN

En este caso se debió crear una capa CNN *input* y sobre la misma ir apilando las demás capas CNN que fueron variando en cantidad desde 1 a 3. Además se ajustaron los hiperparámetros de filtros y tamaño del kernel en cada una de ellas.

```

1   # For first layer
2   for i in range(8, 64, 16):
3       filters_0 = i
4       for j in range(2, 8, 2):
5           kernel_size_0
6
7       # Others layers
8       for i in range(1, 3):
9           cnn_i = stack_cnnlayer(dilation_date=2**i)
10          for j in range(8, 64, 16):
11              filters_i = j
12              for k in range(2, 8, 2):

```

```

13     kernel_size_i = k
14
15     # Model compile
16     for lr in [1e-2, 1e-3, 1e-4]:
17         model_lr = lr
18

```

### CNNRNN

Se realizó una configuración mixta de los hiperparámetros de las otras arquitecturas, dónde la única novedad es la búsqueda en los hiperparámetros de la capa MaxPooling1D.

```

1      # First CNN layer
2      for i in range(8,64,16):
3          filters_0 = i
4          for j in range(2,8,2):
5              kernel_size_0
6
7          # Others CNN layers
8          for i in range(1,3):
9              cnn_i = stack_cnnlayer(dilation_date=2**i)
10             for j in range(8,64,16):
11                 filters_i = j
12                 for k in range(2,8,2):
13                     kernel_size_i = k
14
15             # Pooling layer
16             for i in range(2,8,2):
17                 pool_size = i
18
19             # RNN layer
20             for i in range(8,256,16):
21                 units = i
22
23             # Model compile
24             for lr in [1e-2, 1e-3, 1e-4]:
25                 model_lr = lr
26

```

#### 4.5.3 Resultados *dataset* polución

Ya definido el espacio de búsqueda de los modelos, se procede a la búsqueda del mejor conjunto de hiperparámetros obteniendo la tabla 14 (recordar que se utiliza como métrica MAE). De la misma podemos sacar varias conclusiones interesantes.

- El modelo con la mejor puntuación fue CNNGRU en combinación con el algoritmo *Hyperband*, aunque el segundo lugar (LSTM/Optimización

Bayesiana) le sigue muy de cerca. De hecho todos los modelos están muy parejos.

- Hay una clara tendencia a obtener mejores resultados con combinaciones definidas como **CNNGRU/HB/sig** y **LSTM/BO/tanh**.
- Es interesante notar que a pesar de su arquitectura simple **LSTM** logró batir los resultados de otros modelos.
- Nuestro tope de unidades **RNN** fue seteado en 256, observamos que los modelos con mejor puntaje están muy cerca de ese valor. Existe la posibilidad de obtener mejores resultados en caso de haber definido un espacio de búsqueda mayor para ese hiperparámetro.

model	algo	layers	CNN				MAXPOOL	RNN	DENSE	<b>score</b>
			$f_0$	$ks_0$	$f_1$	$ks_1$				
CNNGRU	HB	3	40	8	56	4	56	6	232	<b>sig</b> 0.011961
LSTM	BO	-	-	-	-	-	-	-	248	<b>tanh</b> 0.011973
CNNGRU	HB	3	40	8	56	4	56	6	232	<b>sig</b> 0.011980
LSTM	BO	-	-	-	-	-	-	-	248	<b>tanh</b> 0.011996
CNNGRU	HB	3	40	8	56	4	56	6	232	<b>sig</b> 0.011996
LSTM	BO	-	-	-	-	-	-	-	248	<b>tanh</b> 0.012001
CNNGRU	HB	1	56	6	-	-	-	-	88	<b>sig</b> 0.012003
LSTM	BO	-	-	-	-	-	-	-	248	<b>tanh</b> 0.012004
CNNGRU	HB	1	40	6	-	-	-	-	120	<b>sig</b> 0.012004
CNNGRU	HB	2	40	6	40	4	-	-	216	<b>sig</b> 0.012008

Cuadro 14: Resultados del ajuste de hiperparámetros.

Se pondrá atención en los rendimientos por arquitectura de ambos algoritmos de ajuste de hiperparámetros utilizados discriminando los análisis por las dos métricas propuestas (**MAE** y **MASE**).

#### MAE

Notamos que exceptuando **LSTM**, el algoritmo *Hyperband* se desempeña mejor en cada uno de los modelos restantes (tabla 15 y fig. 48).

algo/model	CNN	CNNGRU	CNNLSTM	GRU	LSTM
<i>Bayesian Optimization</i>	0.01260	0.01224	0.01222	0.01218	0.01197
<i>Hyperband</i>	0.01213	0.01196	0.01203	0.01204	0.01202

Cuadro 15: Mejor puntuación por algoritmo y arquitectura utilizando **MAE** como métrica.

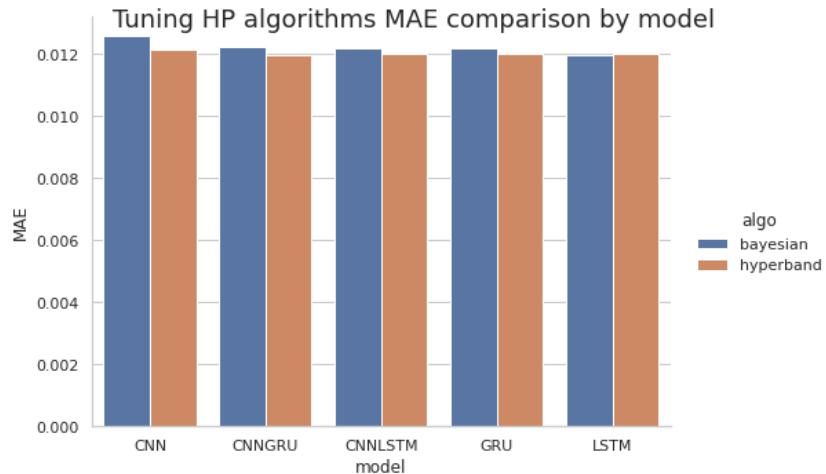


Figura 48: Comparación de algoritmos de ajuste de hiperparámetros por modelo y arquitectura utilizando como métrica MAE.

### MASE

Es arduo crear una función de pérdida MASE ya que ésta recibe como parámetro muestras previas, añadiendo una complejidad que no fue posible superar en su implementación (*Keras* no soportaba esta característica). En consecuencia utilizamos MAE como métrica para entrenar nuestros modelos, sin embargo MASE es nuestra principal métrica a la hora de definir cual es el mejor modelo para el objetivo propuesto.

Concluimos que la combinación que posee el mejor rendimiento en nuestro *set* de datos es CNN con Optimización Bayesiana (tabla 16 y fig. 49).

algo/model	CNN	CNNGRU	CNNLSTM	GRU	LSTM
<i>BayesianOptimization</i>	0,208	0,300	0,306	0,243	0,297
<i>Hyperband</i>	0,253	0,332	0,290	0,628	0,304

Cuadro 16: Mejor resultado por algoritmo y arquitectura utilizando como métrica MASE.

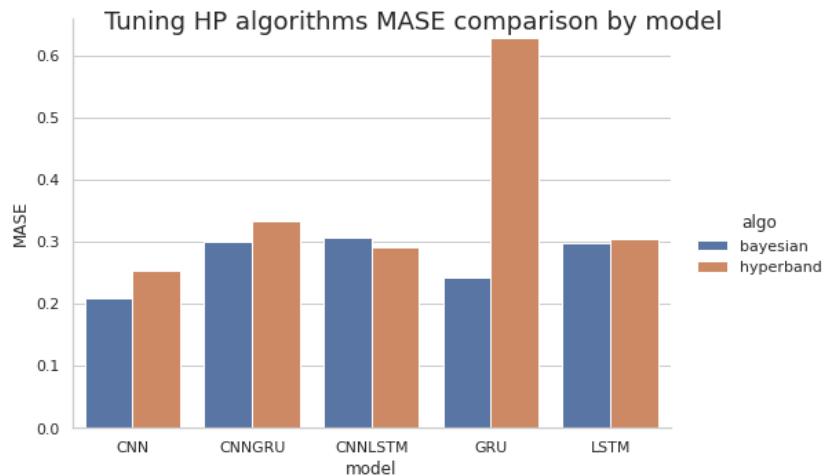


Figura 49: Comparación de algoritmos de ajuste de hiperparámetros por modelo y arquitectura utilizando como métrica MAE.

En la fig. 50 podemos notar la excelente precisión del modelo al predecir los instantes  $t+1$ , justificando de manera visual el análisis realizado.

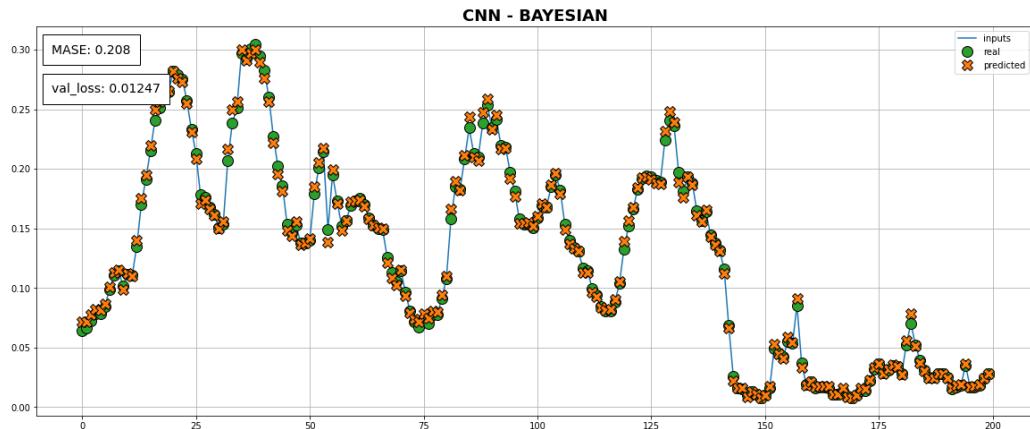


Figura 50: Pronóstico del mejor modelo una vez realizado el ajuste de hiperparámetros.

Para destacar la utilidad del ajuste de hiperparámetros recordemos la tabla 12. En la misma notamos que el mejor modelo (CNNGRU) logró un MASE de 0.278 mientras que el modelo (CNN) de mejor rendimiento una vez aplicado

el ajuste de hiperparámetros obtuvo 0.208, una mejora de casi 8 puntos. También es interesante notar que antes del ajuste de hiperparámetros los modelos CNN estaban muy lejos de los mejores rendimientos, pero luego se posicionaron como la mejor opción.

#### 4.5.4 Conclusión

Si bien el ajuste de hiperparámetros es más costoso computacionalmente que el entrenamiento de los modelos, si la necesidad radica en obtener el mejor modelo posible con nuestros recursos definitivamente es un paso indispensable.

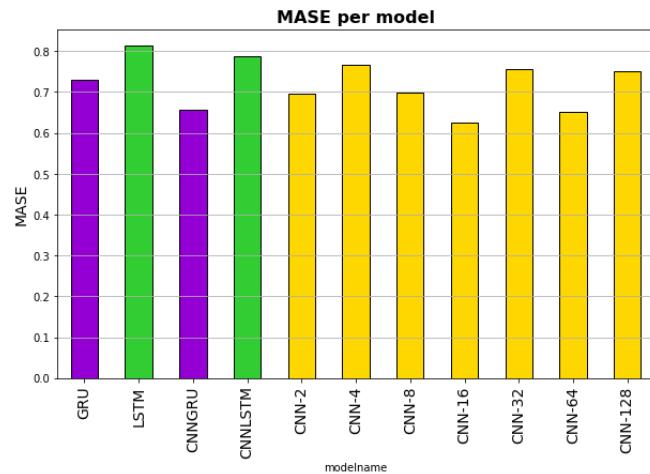
#### 4.5.5 Resultados *dataset FSI*

Los mismos procedimientos realizados sobre nuestra *dummy data* serán aplicados a nuestro *dataset* de FSI.

Los modelos obtenidos sin búsqueda de hiperparámetros arrojan los resultados de la tabla 17 que son interpretados visualmente en la fig. 51. Si comparamos los gráficos entre ambos *datasets*, tenemos un rendimiento 8 veces peor en promedio (MAE) que con el *dataset* de polución, esto claramente repercutirá en la precisión de pronóstico de nuestros modelos.

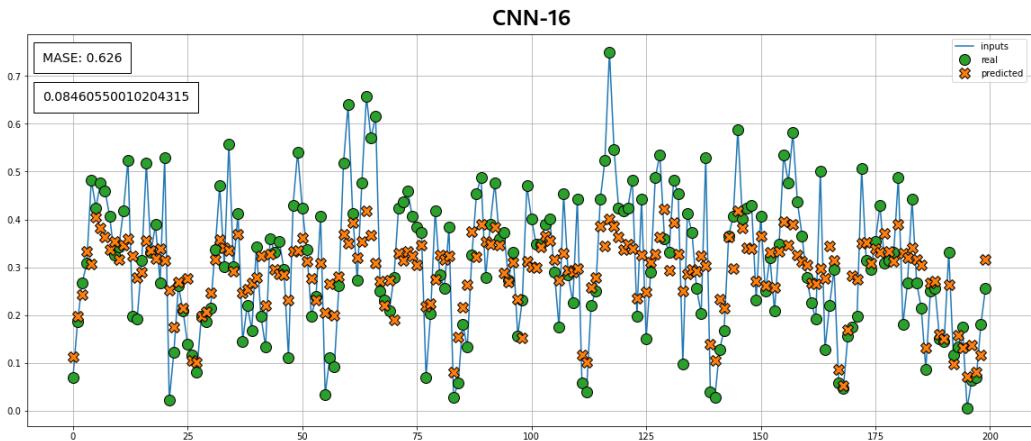
model	val_loss	MASE
GRU	0.0874	0.7308
LSTM	0.0891	0.8138
CNNGRU	0.0865	0.6575
CNNLSTM	0.0841	0.7877
CNN-1	0.0867	0.6812
CNN-2	0.0891	0.6949
CNN-4	0.0888	0.7675
CNN-8	0.0844	0.6984
CNN-16	0.0846	0.6260
CNN-32	0.0839	0.7574
CNN-64	0.0847	0.6516
CNN-128	0.0878	0.7501

Cuadro 17: Métricas de los modelos para el *dataset* FSI.

Figura 51: MASE por modelo con *dataset* FSI.

El mejor modelo con un MASE de 0.626 es CNN-16 seguido de cerca por CNNGRU. Se observa que los modelos LSTM funcionan peor que los demás pero no podemos sacar conclusiones claras aún.

De la fig. 52 notamos que a nuestro modelo no tiene problemas en predecir mínimos pero le es casi imposible predecir máximos debido a los cambios abruptos de la serie temporal.

Figura 52: Mejor modelo para el *dataset* FSI.

Obtenemos la tabla 14 (recordar que se utiliza como métrica MAE). De la misma podemos sacar varias conclusiones interesantes.

- El modelo con la mejor puntuación fue **CNNGRU** en combinación con el algoritmo *HyperBand*, y en segundo lugar (**LSTM/HyperBand**) apenas por detrás.
- 2 capas de CNN fueron suficientes, e incluso podría haberse utilizado solo una.
- Las unidades de las capas RNN varían demasiado sin definir un rango de valores claros.
- Sin lugar a dudas la mejor función de activación fue la sigmoide.

model	algo	layers	CNN				MAXPOOL		RNN	DENSE	score
			$f_0$	$ks_0$	$f_1$	$ks_1$	$f_2$	$ks_2$	$p_s$	units	
CNNGRU	HB	1	56	4	24	6	40	4	6	120	<b>sig</b> 0.080426
CNNGRU	HB	2	56	4	40	4	40	4	6	200	<b>sig</b> 0.080560
CNNLSTM	HB	2	56	2	56	6	24	6	4	168	<b>sig</b> 0.080621
CNNGRU	HB	1	56	6	24	6	56	6	8	184	<b>sig</b> 0.080632
CNNLSTM	HB	1	40	8	24	4	24	6	8	232	<b>sig</b> 0.080693
CNNLSTM	HB	1	24	4	56	4	24	8	6	152	<b>sig</b> 0.080752
CNNLSTM	HB	1	56	6	56	2	56	4	8	248	<b>sig</b> 0.080915
CNNLSTM	HB	1	24	4	40	6	24	4	6	184	<b>sig</b> 0.080979
CNNGRU	HB	2	56	6	24	4	56	4	8	248	<b>sig</b> 0.081028
CNNGRU	HB	1	40	8	40	2	8	6	8	200	<b>sig</b> 0.081080

Cuadro 18: Resultados de ajuste de hiperparámetros del *dataset* FSI.**MAE**

Para esta tarea utilizaremos de soporte la tabla 19 y la fig. 53. Excepto GRU,*Hyperband* obtiene un mejor rendimiento en cada uno de los modelos restantes.

algo/model	CNN	CNNGRU	CNNLSTM	GRU	LSTM
<i>Bayesian Optimization</i>	0.08309	0.08138	0.08138	0.08558	0.08346
<i>Hyperband</i>	0.08196	0.08043	0.08062	0.09082	0.08389

Cuadro 19: Mejor puntuación por algoritmo y arquitectura utilizando MAE como métrica en *dataset* FSI.

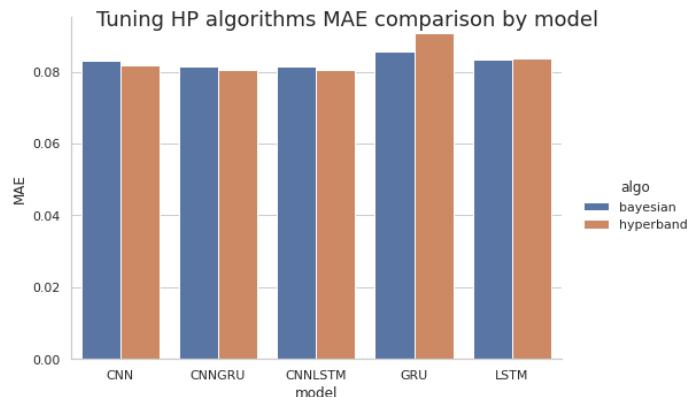


Figura 53: Comparación de algoritmos de ajuste de hiperparámetros por modelo y arquitectura utilizando como métrica MAE.

#### MASE

Utilizando la tabla 20 y la fig. 54 concluimos que la combinación que posee el mejor rendimiento en nuestro *set* de datos es CNN con *HyperBand*. En el *dataset* de polución el mejor rendimiento también se obtuvo a través de una CNN pero el algoritmo de ajuste de hiperparámetros fue *Bayesian*. Aunque en nuestros dos *datasets* hayamos obtenido que el mejor modelo es CNN no se puede asegurar que no habrá arquitecturas que funcionen mejor con otras series temporales.

algo/model	CNN	CNNGRU	CNNLSTM	GRU	LSTM
<i>BayesianOptimization</i>	0.661	0.659	0.707	0.730	0.754
<i>Hyperband</i>	0.627	0.707	0.704	0.744	0.752

Cuadro 20: Mejor resultado por algoritmo y arquitectura utilizando como métrica MASE en *dataset* FSI.

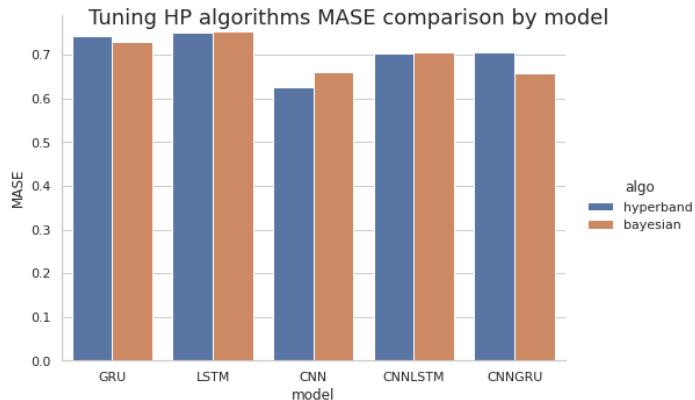


Figura 54: Comparación de algoritmos de ajuste de hiperparámetros por modelo y arquitectura utilizando como métrica MASE para el *dataset* FSI.

En la fig. 55 resaltamos lo antes mencionado, nuestro modelo carece de la capacidad de predecir los valores máximos de la función, pero puede predecir los valores mínimos. Si pensamos a cada capa como un amplificador podemos intuir que quizás una de las mismas se esté saturando incapacitando a las siguientes a llegar a los valores máximos. Se realizará un análisis ajustando los datos de entrada al modelo con diferentes *scalers*.

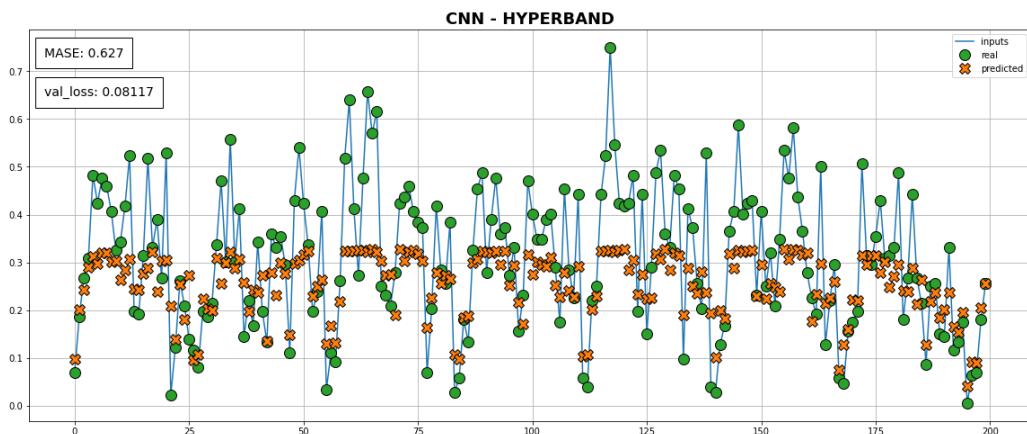


Figura 55: Pronóstico del mejor modelo para el *dataset* FSI una vez realizado el ajuste de hiperparámetros.

Este inconveniente lo hace disfuncional a nuestra tarea pero en la próxima sección intentaremos descubrir que diferencia a ambos *datasets* para que uno

funcione tan bien y el otro no.

Comparemos los valores antes y después del ajuste de hiperparámetros, en la tabla 17 obtuvimos con CNN-16 un MASE de 0.626 y la tabla 20 nos dice que el mejor modelo fue un CNN con MASE de 0.627 *i.e.* valores casi idénticos pero además peores para el post-ajuste.

#### 4.5.6 Cambio en preprocessamiento de datos

Como mencionamos anteriormente, para evitar la saturación en nuestra red variaremos forma en que la misma recibe nuestro *dataset* modificando dos parámetros:

- *Scaler*: algoritmo de preprocessamiento de datos cuyo objetivo usualmente es lograr un *dataset* más homogéneo y sin *outliers*.
- Factor de escala ( $s_f$ ): simplemente multiplicaremos toda la red por este valor para obtener amplitudes menores.

*Scikit-learn* nos provee de varios *scalers* pero seleccionamos los siguientes:

- **MinMax**: escala cada característica individualmente de modo que se encuentre en el rango seleccionada (*e.g.* 1 y 0).
- **PowerTransformer**: aplica una transformación de potencia en función de las características para hacer que los datos sean más *gaussianos*.
- **RobustScaler**: escala características usando estadísticas que sean robustas a *outliers*.
- **StandardScaler**: estandariza las características eliminando la media y escalando a la varianza unitaria.

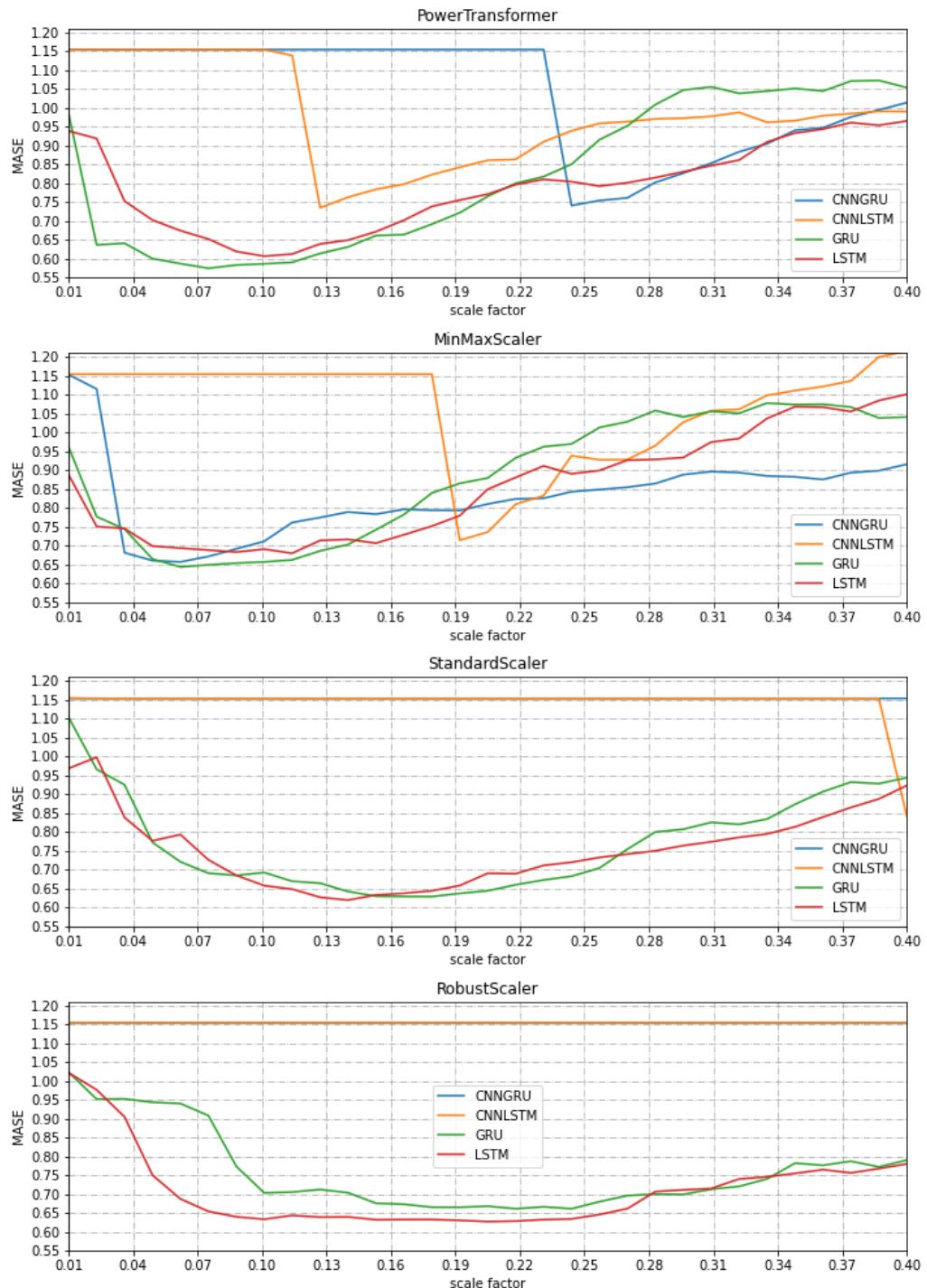
Para el factor de escala se definió un rango entre 0.01 y 0.40 tras varios ensayos.

En la tabla 21 se corrobora que el *scaler* de mejor rendimiento es **PowerTransformer** (notar los bajos valores de  $s_f$ ). Con respecto a nuestros resultados originales (en MASE) se obtienen casi 5 puntos de mejora.

<i>scaler</i>	$s_f$	model	MASE
PowerTransformer	0.075	GRU	0.5749
PowerTransformer	0.088	GRU	0.5836
PowerTransformer	0.101	GRU	0.5868
PowerTransformer	0.062	GRU	0.5873
PowerTransformer	0.114	GRU	0.5907
PowerTransformer	0.049	GRU	0.6004
PowerTransformer	0.101	LSTM	0.6070
PowerTransformer	0.114	LSTM	0.6128
PowerTransformer	0.127	GRU	0.6142
PowerTransformer	0.088	LSTM	0.6194

Cuadro 21: Top 10 de mejores *scalers* según MASE obtenido.

Los modelos RNN puros funcionan mucho mejor que los que poseen algún componente CNN (fig. 56), una posibilidad es que las capas CNN se sobrecarguen haciendo que el modelo diverja. Además según el *scaler* hay determinadas zonas de valle en las cuales los modelos funcionan mejor. Lo recomendable sería una vez determinado el mejor par *scaler*-zona de mayor rendimiento, explorar utilizando ajuste de hiperparámetros.

Figura 56: Factor de escala *vs* MASE según *scaler*.

Los resultados son esclarecedores, si bien seguimos sin llegar a los picos ahora al menos nos aproximamos (fig. 57) y no tenemos ese techo que no nos permitía escalar (fig. 55). Otra manera de mejorar el rendimiento de nuestros modelos puede ser variar las configuraciones del preprocesamiento de los datos. Sin embargo esto es tedioso y arduo de automatizar.

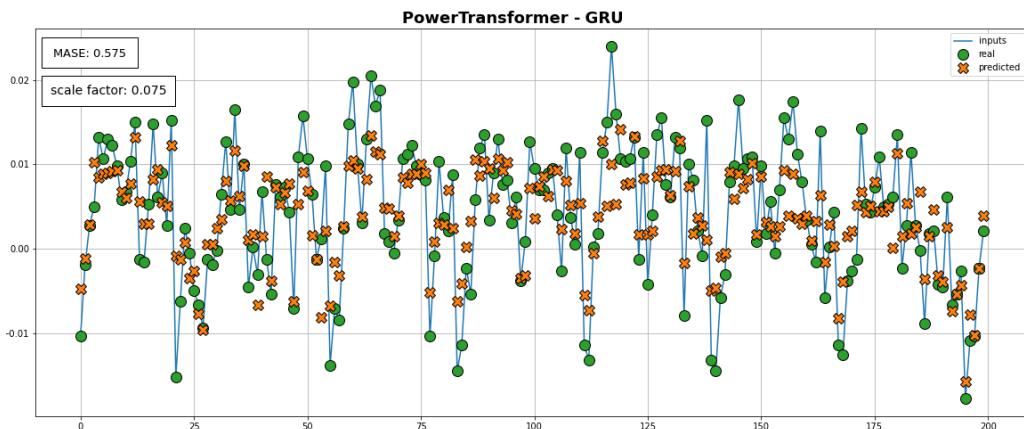


Figura 57: Prónostico utilizando como *scaler* PowerTransformer y modelo GRU.

#### 4.5.7 Conclusión

No siempre aplicar ajuste de hiperparámetros nos dará resultados mejores a los originales, ya que esto dependerá de cada serie temporal a pronosticar. Pero si intuimos lo que puede estar sucediendo en la red podemos desarrollar soluciones innovadoras.

### 4.6 Entropía

Intentaremos encontrar una causa al dispar rendimiento entre nuestro *dataset* de polución del aire y el de defectos. Para ello haremos uso de lo revisado en la sección ??, dónde mencionamos que la entropía de los datos influye directamente en la capacidad de predicción de nuestros modelos. Mientras más entropía tengan los datos, más difícil será que el modelo pueda aprender correctamente sus patrones, si es que los hubiere.

Ambos *datasets* serán procesados por los dos algoritmos disponibles para esta tarea: ApEn y SampEn.

#### 4.6.1 Rendimiento de los algoritmos

Al realizar los ensayos quedó al descubierto la demora de los algoritmos para ejecutar los cálculos sobre la totalidad del *dataset*, lo que llevó a buscar una solución. *Python* por defecto no posee librerías que hagan uso intensivo de GPU (un caso es *scikit-learn*, que a pesar de brindar diversos algoritmos de Aprendizaje Automático sólo utiliza CPU para sus cálculos) y *NumPy* (librería por defecto para operaciones matriciales y tensoriales) no es la excepción.

##### CuPy

Es una biblioteca de matrices de código abierto que proporciona computación acelerada por GPU con *Python* a través de CUDA. [60]

CUDA es una plataforma de computación paralela y un modelo de programación desarrollado por *Nvidia* para computación general en unidades de procesamiento gráfico (GPU). Con CUDA, los desarrolladores pueden acelerar drásticamente las aplicaciones informáticas aprovechando la potencia de las GPU. [61] La implementación de CuPy sobre código creado para la librería NumPy es muy simple.

##### Comparación de rendimiento

Se divisa que a medida que la cantidad de muestras aumenta, el tiempo de ejecución de NumPy crece exponencialmente en contraste con CuPy que crece de forma lineal (fig. 58). En el caso límite que el algoritmo utiliza la totalidad del *dataset* llegamos a tener un *speedup* de 8X. Por tanto concluimos que CuPy promete lo que cumple.

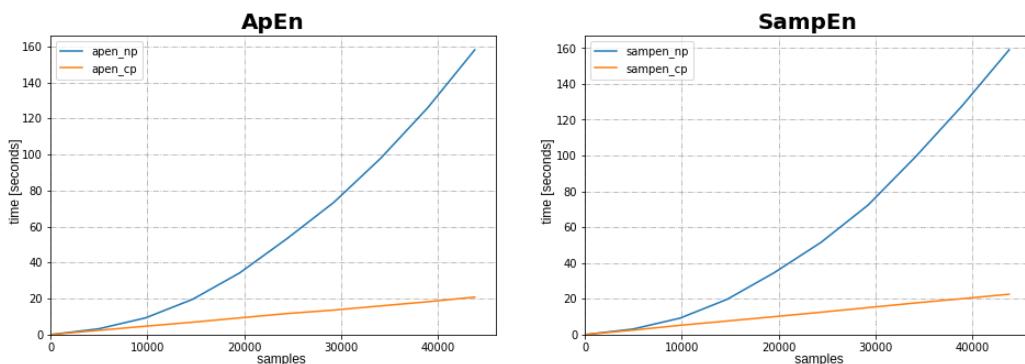


Figura 58: Comparación de NumPy *vs* CuPy.

#### 4.6.2 *Dataset de polución*

Vamos a calcular ApEn y SampEn sobre el *dataset* de polución con  $0 \leq m \leq 4$  y  $0 \leq r \leq 0.25$  (recordar que éste valor debe ser multiplicado por la desviación estándar de la serie temporal  $\sigma$ ). Los resultados obtenidos son visualizados en la fig. 59.

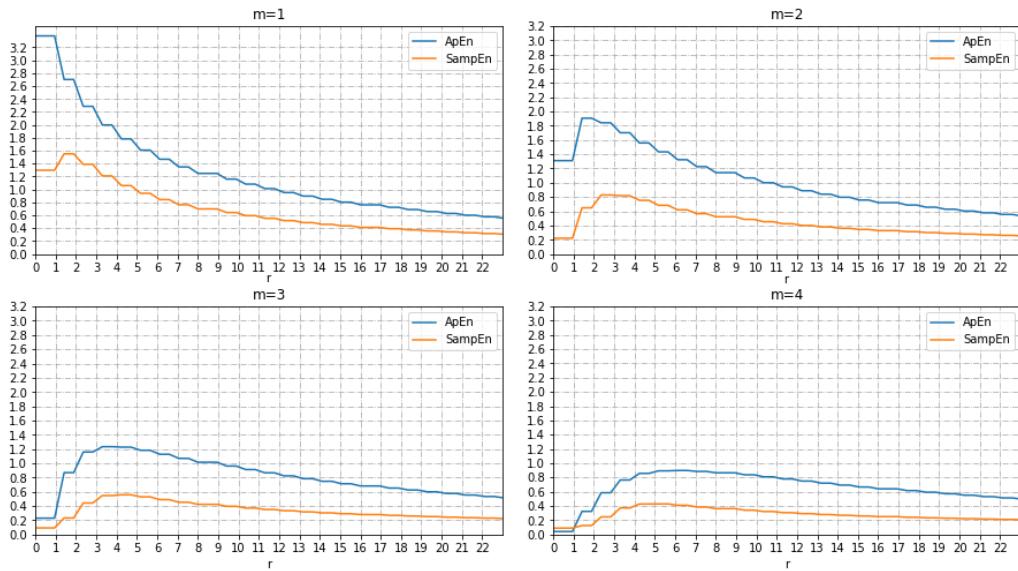


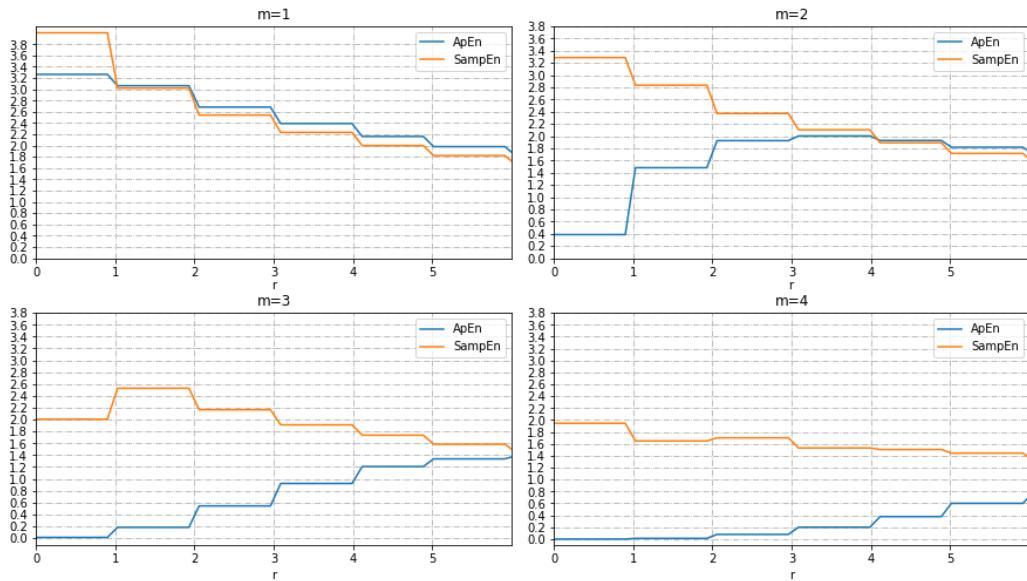
Figura 59: Gráfico de la entropía del *dataset* de polución.

Podemos deducir que tenemos un pico de entropía claro en cada uno de las figuras, que a medida que  $m$  aumenta este pico se va desplazando hacia la derecha aumentando el valor de  $r$  que lo intersecta.

#### 4.6.3 *Dataset FSI*

Sin embargo, esta información carece de valor si no la comparamos con nuestro *dataset* principal, por tanto observemos los resultados obtenidos para el mismo en la fig. 62.

La forma escalonada se debe a la baja densidad de datos en comparación al otro *dataset*. Si bien se divisa que los valores de ApEn y SampEn a priori parecen mayores, no podemos sacar conclusiones claras por lo que se requiere otro tipo de análisis.

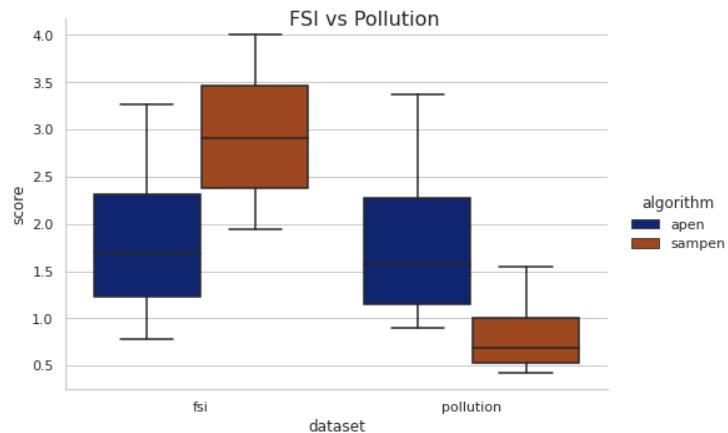
Figura 60: Gráfico de la entropía del *dataset* FSI.

#### 4.6.4 Comparación de *datasets*

Tomaremos el mayor resultado de cada  $m$  de cada modelo y algoritmo, conformando la tabla 22. Interpretaremos estos valores a través de la fig. 62, que nos muestra que según ApEn ambas series temporales muestran una entropía similar, cosa que podemos desmentir al visualizar los ploteos de cada una.

Sin embargo, SampEn nos ayuda a exponer de manera empírica que la entropía de la serie temporal de FSI es mayor a la de polución. En este caso SampEn funcionó mejor que ApEn a la hora de comparar la entropía de ambas series temporales.

m	score	dataset	algo
1	3.264	fsi	ApEn
2	2.003	fsi	ApEn
3	1.388	fsi	ApEn
4	0.777	fsi	ApEn
1	4.003	fsi	SampEn
2	3.286	fsi	SampEn
3	2.527	fsi	SampEn
4	1.946	fsi	SampEn
1	3.377	pollution	ApEn
2	1.906	pollution	ApEn
3	1.233	pollution	ApEn
4	0.897	pollution	ApEn
1	1.551	pollution	SampEn
2	0.828	pollution	SampEn
3	0.558	pollution	SampEn
4	0.427	pollution	SampEn

Cuadro 22: Tabla para comparación de *datasets*.Figura 61: Gráfico de la entropía del *dataset* FSI.

## 5 Despliegue

El objetivo de crear un modelo de aprendizaje automático es resolver un problema. Esto solo es posible cuando está en producción y los consumidores lo utilizan activamente. Como tal, el despliegue de modelos es tan importante como su construcción.

Los científicos de datos se destacan en la creación de modelos que representan y predicen datos del mundo real, pero la implementación efectiva de modelos de aprendizaje automático requiere otro tipo de habilidades se encuentran más comúnmente en ingeniería de software y *DevOps*. El 87% de los proyectos de ciencia de datos nunca llegan a producción, mientras que otras fuentes afirman que es el 90%. [62] Un factor crítico que marca la diferencia entre el éxito y el fracaso es lograr capitalizar el valor de nuestros modelos.

Las dificultades en la implementación y gestión de modelos han dado lugar a un nuevo rol especializado: el ingeniero de *machine-learning* o *ML-Ops*. Estos especialistas están más cerca de los ingenieros de *software* que de los científicos de datos y son el candidato ideal para poner modelos en producción.

### 5.1 Áreas fundamentales a tener en cuenta para un proyecto de inteligencia artificial

Las tres áreas claves que un equipo debe considerar antes de embarcarse en cualquier proyecto de inteligencia artificial son las siguientes:

- Almacenamiento y extracción de datos.
- *Frameworks* y herramientas.
- *Feedback* e iteración.

#### 5.1.1 Almacenamiento y extracción de datos

##### Almacenamiento de datos

Los datos se pueden almacenar en los servidores locales, en la nube o en un híbrido de ambos. Es sensato almacenar los datos donde ocurrirá el entrenamiento del modelo y se entregarán los resultados.

##### Tamaño de los datos

Si su *dataset* es grande, entonces necesita más potencia de cálculo para los pasos de preprocesamiento y las fases de optimización del modelo. Esto significa que debe planificar una mayor capacidad de procesamiento si opera

localmente o configurar el escalado automático en un entorno de nube desde el principio.

#### **Extracción de datos para entrenamiento**

Otro aspecto importante que debe considerarse antes de diseñar el sistema de *machine-learning* es la extracción y procesamiento de datos esto para el entrenamiento del modelo.

- **Por lotes:** los datos se extraen en trozos de un sistema de almacenamiento.
- **En tiempo real:** los datos se extraen tan pronto como están disponibles.

#### **Extracción de datos para inferencia**

Se toman los mismos enfoques revisados en el punto anterior.

- **Por lotes:** Usualmente se almacena la solicitud de predicción y luego se realiza la inferencia según la carga de trabajo del servidor.
- **En tiempo real:** La inferencia se realiza tan pronto como se realiza la solicitud de predicción, por consiguiente se necesita disponibilidad inmediata de recursos.

#### **5.1.2 *Frameworks* y herramientas**

Después de examinar y preparar su *pipeline* de datos, la siguiente línea de pensamiento debe tener en cuenta la combinación de *frameworks* y herramientas utilizar. La elección del *framework* es muy importante, ya que puede decidir la continuidad, el mantenimiento y el uso de un modelo.

Para ayudar a determinar la mejor herramienta para la tarea, debe investigar y comparar las características de diferentes herramientas que realizan el mismo trabajo.

Luego es posible comparar estas herramientas según criterios tales como:

- **Eficiencia:** Un marco o herramienta es eficiente si utiliza de manera óptima recursos como la memoria, la CPU o el tiempo. Es importante considerar la eficiencia de los *frameworks* o las herramientas que pretende utilizar porque tienen un efecto directo sobre el rendimiento, la confiabilidad y la estabilidad del proyecto.
- **Popularidad:** este parámetro a menudo significa que funciona bien, que está en uso activo y posee apoyo de *sponsors* importantes.
- **Soporte:** este es un punto crítico ya que impactará de forma directa en los tiempos de resolución de problemas por parte del equipo. Se debe evaluar si la librería tiene una comunidad activa en caso de ser código abierto, o si tiene un buen soporte si es código cerrado.
- **Plataformas:** debe seleccionar en qué tipo de plataformas se encontrará disponible su modelo. Algunas de ellas son plataformas móviles

(*Android* ó *iOS*), de escritorio (*Windows*, *Linux* ó *Mac*) o web (navegador).

### 5.1.3 ***Feedback*** e iteración

Los proyectos de inteligencia artificial nunca son estáticos, esto es un aspecto del diseño que debe considerarse desde sus bases.

#### ***Feedback***

El seguimiento activo y la supervisión del estado del modelo en producción pueden advertirle en casos de depreciación o deterioro del rendimiento del modelo. Esto garantizará la toma de acciones tempranas para evitar consecuencias a largo plazo.

#### **Iteración**

Se debe considerar cómo experimentar, reentrenar e implementar nuevos modelos en producción sin interrumpir el funcionamiento de ese modelo. Un nuevo modelo debe probarse adecuadamente antes de reemplazar el anterior. Esta idea de prueba continua y despliegue de nuevos modelos sin interrumpir los procesos del modelo existente se denomina **integración continua**.

## 5.2 Análisis del problema

Considerando los puntos expuestos en la sección 5.1 analizaremos nuestro problema.

### 5.2.1 Almacenamiento y extracción de datos

#### **Almacenamiento de datos**

La fábrica almacena sus datos en servidores en la nube pero el proveedor es *SAP*, que no proporciona de ningún tipo de automatización para la consulta de datos siendo este un gran impedimento para la implementación del modelo propuesto. Se debería crear un puente entre esta base de datos y el modelo, costos que la compañía no está dispuesta a afrontar.

#### **Tamaño de los datos**

Se desconoce el tamaño total de los datos debido a que no se provee acceso directo a la base de datos. En consecuencia se tomo una muestra que abarca el período 2020-2021.

#### **Extracción de datos para entrenamiento**

Este proceso debió realizarse de forma manual ya que como se detalló anteriormente no hay acceso a la base de datos, menos en tiempo real. No obstante, si proponemos una situación ideal este modelo necesita disponer de los datos en tiempo real.

### Extracción de datos para inferencia

Al igual que en el punto anterior, se necesita tener a disposición los datos en tiempo real para obtener la predicción continua del modelo.

#### 5.2.2 *Frameworks* y herramientas

Es punto se revisará en la sección 5.3.

#### 5.2.3 *Feedback* e iteración

Ya en la sección 5.2.1 se reconoció la inviabilidad de la puesta en producción del modelo. Por tanto no es necesario considerar estos puntos.

### 5.3 Arquitectura propuesta

Sin embargo, consideremos una situación ideal dónde los inconvenientes enumerados en la sección son sorteados. En tal situación expondremos una posible implementación de nuestro modelo de forma teórica.

#### 5.3.1 *Frameworks*, herramientas y *software*

La arquitectura propuesta se muestra en la figura 62. En esta sección se irá desglosando y explicando cada uno de sus componentes mostrados, además de su relación entre ellos.[63]

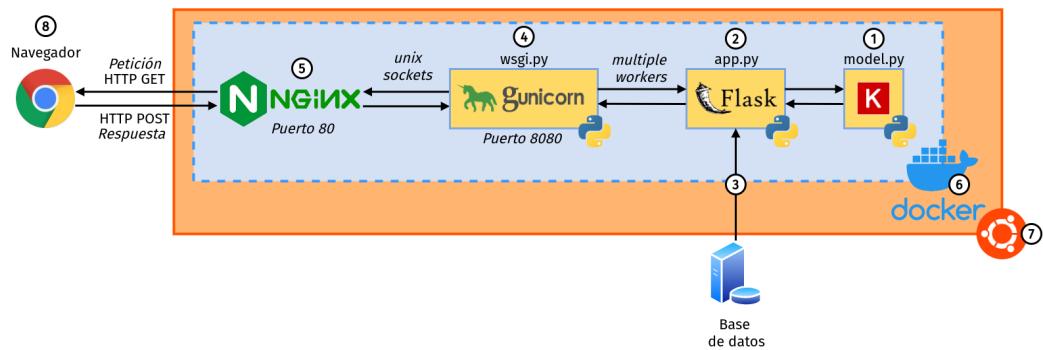


Figura 62: Arquitectura de despliegue para modelo de *machine-learning*.

- **Flask:** es un *framework* web y módulo de *Python* que permite desarrollar aplicaciones web fácilmente. [64]

- **Gunicorn:** un servidor HTTP WSGI de *Python*. Es ampliamente compatible con varios *frameworks* web, implementado de manera simple, ligero en recursos del servidor y bastante rápido. [65]
- **Nginx:** software de código abierto para servicio web, proxy inverso, almacenamiento en caché, equilibrio de carga, transmisión de medios y más. [66]
- **Ubuntu:** sistema operativo de escritorio y servidores *Linux* basado en la arquitectura y la infraestructura de *Debian*. [67]
- **Docker:** es una herramienta que facilita el empaquetado de aplicaciones en contenedores autosuficientes. Los contenedores contienen todas las partes que la aplicación necesita para ejecutarse, desde bibliotecas y dependencias hasta el código fuente real. [68]

### 5.3.2 Descripción

Procedemos a describir cada una de las etapas de la arquitectura:

1. En `model.py` estará el código necesario de nuestro modelo contenido en la clase `Model()`. Esta clase se sirve del método `predict()` para inferir, tomando como entrada datos de  $M$  columnas por  $N$  pasos y retornando el dato  $N + 1$ .
2. En `app.py` crearemos una instancia de nuestro modelo y con la librería `Flask` definiremos la ruta a nuestro método `predict()`.
3. Creamos una conexión entre la base de datos y `app.py`, recordar que el flujo de datos debe ser en tiempo real.
4. `WSGI` son las siglas de *Web Server Gateway Interface* y es el intermedio entre su aplicación `Flask` y su servidor web (que se configurará mediante `Nginx`). `Gunicorn` es una de opciones de `WSGI` que podemos configurar y usar fácilmente con `Flask` (el `WSGI` integrado que posee no está diseñado para la producción). En `wsgi.py` instanciaremos `Flask`, además de permitirnos configurar algunos parámetros como el puerto, el `time-out` y la cantidad de trabajadores en simultáneo.
5. Configuramos nuestro servidor web `Nginx` para mapear el puerto 80 al 8080 de modo que el servidor web actué como un *proxy* y `Gunicorn` reciba y procese las solicitudes.
6. Se debe crear una imagen de `Docker` con todos los archivos y dependencias necesarias para correr nuestra aplicación. Esto garantiza que si migramos nuestro modelo a otro sistema operativo ó *hardware* seguirá funcionando, además de facilitar el mantenimiento del mismo.

7. El sistema operativo donde vamos a ejecutar nuestra aplicación será *Linux*, más precisamente en la distribución *Ubuntu*. Las razones de la decisión son el mantenimiento y confiabilidad que nos brinda (una de las premisas de la distribución es la estabilidad y realiza lanzamientos en promedio cada dos años).
8. En última instancia la predicción puede ser visualizada a través de una página web de un navegador. La idea es que el personal sea reactivo a la hora de atacar los defectos, al detectarlos antes que sucedan.

## 6 Conclusiones

El proceso de construcción del sistema de predicción de defectos, requirió el estudio y selección de las técnicas de inteligencia artificial apropiadas para la ejecución del proyecto, las cuales luego de un cuidadoso estudio se identificaron que las más aptas eran redes neuronales de dos tipos: convolucionales y recurrentes.

La recolección de datos significó una ardua tarea ya que los mismos se encontraban dispersos en diversas fuentes. Con el análisis de datos, se identificaron que se montaron un total de 10780 vehículos, con una media de 19.6 defectos. Del total un 14.7% pertenece a defectos graves y el resto a defectos leves. Por otro lado, se identificó que el departamento con mayor cantidad de defectos con más del 50% es **PINTURA** y dentro de las familias fue **ASPECTO** con casi un 70%. Por lo cual se distingue una gran correlación entre estas dos variables.

La mayor proporción de defectos graves fue en la combinación **MONTAJE-FALTANTES**, denotando que la cantidad de defectos no está íntimamente relacionado con la gravedad de los mismos.

Las técnicas de inteligencia artificial evaluadas originalmente si bien no fueron seleccionadas para el modelo definitivo, fueron de utilidad para imputar datos faltantes en las columnas **TIPO** y **UET** con una precisión del 99.73% y 79.74% respectivamente.

Se especificó la arquitectura para cada modelo y las métricas para su evaluación. Cada modelo se entrenó con datos de prueba y datos objetivo para contrastar y verificar su correcta ejecución. El mínimo error obtenido para los datos de prueba utilizando como métrica **MASE** arrojó como resultado 27.75% para un modelo convolucional-recurrente. Visualmente se constata que el modelo es capaz de predecir con exactitud los valores objetivo. Para los datos objetivo el error crece a 62.6% siendo esto reflejado en los gráficos.

Se aplicó búsqueda de hiperparámetros a cada modelo para acceder a la máxima eficacia posible, limitando cada configuración a un espacio de búsqueda predefinido. El modelo convolucional-causal a través de hiperparámetros obtenidos mediante búsqueda bayesiana disminuyó el error a 20.80% en los datos de prueba.

Los modelos para los datos objetivo pese a que diversas estrategias fueron puestas a prueba en el preprocesamiento de datos solo lograron reducir el error en un pequeño porcentaje. El modelo recurrente con un *scaler PowerTransformer* obtuvo el mejor desempeño con un error equivalente a 57.5%.

Si bien los modelos de redes neuronales son potentes para el pronóstico de series temporales, es factible que su rendimiento se vea afectado por la calidad de los datos. Un modo de estimar este aspecto es a través del cómputo de la entropía de la serie temporal, que nos brinda indicios de la condición de nuestros datos sin preocuparse por su origen. Consecuentemente se comprobó que la entropía en los datos objetivo era superior a la de los datos de prueba.

En el despliegue propuesto del modelo se revisó cada una de las etapas que componen un sistema de inteligencia artificial. Así tomamos dimensión que implementar un sistema de esta envergadura requiere de diversos profesionales a fin de poner en marcha cada uno de los procesos necesarios. Se requiere de una profunda re-estructuración a nivel técnico, por lo cual la compañía realizará un análisis de la viabilidad del proyecto para proporcionar los fondos requeridos.

Para posibles trabajos futuros se sugiere tener en cuenta las siguientes recomendaciones:

1. Explorar en mayor profundidad el preprocesamiento de los datos constatando la repercusión de estos cambios en el desempeño del modelo.
2. Si la organización lo requiere, incluir metadata externa a nuestros datos originales para así obtener un *dataset* más diverso.
3. Experimentar con otras herramientas *open-source* disponibles para el pronóstico de series temporales.

Desde un punto de vista funcional, los resultados han sido satisfactorios para el comitente, ya que los modelos cuentan con rendimientos adecuados para la naturaleza del problema abordado. En lo personal todo el recorrido a través del campo *Machine Learning* ha sido muy satisfactorio, adquiriendo habilidades sobre herramientas muy valoradas profesionalmente.

## Referencias

- [1] Ian Sommerville. Software engineering 10th edition. *ISBN-10*, 137035152:18, 2015.
- [2] ¿Qué es el Principio de Pareto? La Ley del 80/20 | Blog de Anfix, Nov 2021. URL <https://www.anfix.com/blog/que-es-el-principio-de-pareto>. [Online; accessed 11. Nov. 2021].
- [3] David I Poole, Randy G Goebel, and Alan K Mackworth. *Computational intelligence*. Oxford University Press New York, 1998.
- [4] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.
- [5] A.T. Norman and S. Bolivar. *Aprendizaje automático en acción. Un libro para el lego, guía paso a paso para los novatos*. Tektme, 2019.
- [6] Pedro Antonio Gutiérrez. Github - pagutierrez/tutorial-sklearn: Tutorial sobre scikit-learn completo. <https://github.com/pagutierrez/tutorial-sklearn>, 2020. (Accessed on 12/28/2020).
- [7] A. Rosebrock. *Deep Learning for Computer Vision with Python: Starter Bundle*. PyImageSearch, 2017.
- [8] D. J. Matich. *Redes neuronales: Conceptos básicos y aplicaciones*. Universidad Tecnológica Nacional, FRR, Departamento de Ing. Química, 2001.
- [9] Colaboradores de los proyectos Wikimedia. Soma (neurología) - Wikipedia, la enciclopedia libre, May 2021. URL [https://es.wikipedia.org/w/index.php?title=Soma\\_\(neurolog%C3%ADA\)&oldid=135426685](https://es.wikipedia.org/w/index.php?title=Soma_(neurolog%C3%ADA)&oldid=135426685). [Online; accessed 26. Oct. 2021].
- [10] Ph. D. Ayyüce Kızrak. Comparison of Activation Functions for Deep Neural Networks. *Medium*, Jan 2020. ISSN 7064-2848. URL <https://towardsdatascience.com/comparison-of-activation-functions-for-deep-neural-networks-706ac4284c8a>.
- [11] freeCodeCamp.org. Demystifying gradient descent and backpropagation via logistic regression based image...,

- Jul 2018. URL <https://www.freecodecamp.org/news/demystifying-gradient-descent-and-backpropagation-via-logistic-regression-based-on-a-real-world-example/>
- [12] Wikipedia. Gradient descent - wikipedia. [https://en.wikipedia.org/wiki/Gradient\\_descent#An\\_analogy\\_for\\_understanding\\_gradient\\_descent](https://en.wikipedia.org/wiki/Gradient_descent#An_analogy_for_understanding_gradient_descent), 2020. (Accessed on 12/31/2020).
- [13] Quora. What are the key trade-offs between overfitting and underfitting?, 2020. URL <https://www.quora.com/What-are-the-key-trade-offs-between-overfitting-and-underfitting>.
- [14] Frank Keller. *Convolutions and Kernels*. School of Informatics, University of Edinburgh, Feb 2010.
- [15] Cognethi. C 4.1 | 1D Convolution | CNN | Object Detection | Machine Learning | EvODN, Aug 2019. URL [https://www.youtube.com/watch?v=yd\\_j\\_zdLDWs](https://www.youtube.com/watch?v=yd_j_zdLDWs). [Online; accessed 4. Jan. 2021].
- [16] Louis N Andrianaivo, Roberto D'Autilia, and Valerio Palma. Architecture recognition by means of convolutional neural networks. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, 2019.
- [17] Sumit Saha. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. *Medium*, Oct 2020. ISSN 3211-6453. URL <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b116>
- [18] 6.3. Padding and Stride — Dive into Deep Learning 0.16.0 documentation, Jan 2021. URL [https://d2l.ai/chapter\\_convolutional-neural-networks/padding-and-strides.html](https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html). [Online; accessed 11. Jan. 2021].
- [19] StackOverflow. How did they calculate the output volume for this convnet example in Caffe?, Jan 2021. URL <https://stackoverflow.com/questions/32979683/how-did-they-calculate-the-output-volume-for-this-convnet-example-in-caffe>. [Online; accessed 4. Jan. 2021].
- [20] Federico Peccia. Batch normalization: theory and how to use it with Tensorflow. *Medium*, Jan 2019. ISSN 1892-0173. URL <https://towardsdatascience.com/batch-normalization-theory-and-how-to-use-it-with-tensorflow-1892ca0173ad>.

- [21] Deepmind. WaveNet: A Generative Model for Raw Audio, Sep 2016. URL <https://deepmind.com/blog/article/wavenet-generative-model-raw-audio>. [Online; accessed 11. Jan. 2021].
- [22] Joseph Eddy. Time Series Forecasting with Convolutional Neural Networks - a Look at WaveNet, Feb 2019. URL [https://jeddy92.github.io/JEddy92.github.io/ts\\_seq2seq\\_conv](https://jeddy92.github.io/JEddy92.github.io/ts_seq2seq_conv). [Online; accessed 9. Jan. 2021].
- [23] Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks, Jun 2020. URL <https://karpathy.github.io/2015/05/21/rnn-effectiveness>. [Online; accessed 3. Jan. 2021].
- [24] Christopher Olah. Understanding LSTM Networks, Aug 2015. URL <https://colah.github.io/posts/2015-08-Understanding-LSTMs>. [Online; accessed 4. Jan. 2021].
- [25] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019. ISBN 9781492032595.
- [26] Michael Phi. Illustrated Guide to Recurrent Neural Networks - Towards Data Science. *Medium*, Sep 2019. ISSN 7958-0499. URL <https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9>.
- [27] Producto de Hadamard (matrices) - Hadamard product (matrices) - qaz.wiki, Jan 2021. URL [https://es.qaz.wiki/wiki/Hadamard\\_product\\_\(matrices\)](https://es.qaz.wiki/wiki/Hadamard_product_(matrices)). [Online; accessed 4. Jan. 2021].
- [28] Will Koehrsen. *A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning*. Towards Data Science, India, Jul 2018. ISBN 978-817227805. URL <https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization>
- [29] Andre Ye. The Beauty of Bayesian Optimization, Explained in Simple Terms. *Medium*, Oct 2020. URL <https://towardsdatascience.com/the-beauty-of-bayesian-optimization-explained-in-simple-terms-81f3ee13b10f>.
- [30] Colaboradores de los proyectos Wikimedia. Vientre de alquiler (práctica) - Wikipedia, la enciclopedia libre, Mar 2021. URL [https://es.wikipedia.org/w/index.php?title=Vientre\\_de\\_alquiler\\_\(pr%C3%A1ctica\)&oldid=134152025](https://es.wikipedia.org/w/index.php?title=Vientre_de_alquiler_(pr%C3%A1ctica)&oldid=134152025). [Online; accessed 4. Apr. 2021].

- [31] Apoorv Agnihotri and Nipun Batra. Exploring Bayesian Optimization. *Distill*, 5(5):e26, May 2020. ISSN 2476-0757. doi: 10.23915/distill.00026.
- [32] Wei Wang. Bayesian Optimization Concept Explained in Layman Terms. *Medium*, Apr 2020. URL <https://towardsdatascience.com/bayesian-optimization-concept-explained-in-layman-terms-1d2bcdeaf12f>.
- [33] James Bergstra, R. Bardenet, Balázs Kégl, and Y. Bengio. Algorithms for hyper-parameter optimization. 12 2011.
- [34] BOHB: Robust and Efficient Hyperparameter Optimization at Scale, Apr 2021. URL [https://www.automl.org/blog\\_bohb](https://www.automl.org/blog_bohb). [Online; accessed 5. Apr. 2021].
- [35] Aloïs Bissuel. *Hyper-parameter optimization algorithms: a short review*. Criteo R&D Blog, Apr 2019. ISBN 978-244752590. URL <https://medium.com/criteo-engineering/hyper-parameter-optimization-algorithms-2fe447525903>.
- [36] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018.
- [37] Noam (xn-45q. xn-9kq. ) Rosenberg. Hyper Parameter Tuning — A Tutorial - Towards Data Science. *Medium*, Aug 2020. ISSN 7065-5254. URL <https://towardsdatascience.com/hyper-parameter-tuning-a-tutorial-70dc6c552c54>.
- [38] Alexandre Abraham. A (Slightly) Better Budget Allocation for Hyperband | data from the trenches. *Medium*, Feb 2021. URL <https://medium.com/data-from-the-trenches/a-slightly-better-budget-allocation-for-hyperband-bbd45af14481>.
- [39] Shubham Agrawal. Time Series Modeling, Mar 2019. URL [https://rstudio-pubs-static.s3.amazonaws.com/478704\\_10185309918043d4a279e1e1545694e0.html](https://rstudio-pubs-static.s3.amazonaws.com/478704_10185309918043d4a279e1e1545694e0.html). [Online; accessed 5. Apr. 2021].
- [40] Prophet, Jan 2021. URL <https://facebook.github.io/prophet>. [Online; accessed 8. Feb. 2021].
- [41] GluonTS - Probabilistic Time Series Modeling — GluonTS documentation, May 2020. URL <https://ts.gluon.ai>. [Online; accessed 8. Feb. 2021].

- [42] alan-turing institute. sktime, Feb 2021. URL <https://github.com/alan-turing-institute/sktime>. [Online; accessed 8. Feb. 2021].
- [43] Alfonso Delgado-Bonal and Alexander Marshak. Approximate entropy and sample entropy: A comprehensive tutorial. *Entropy*, 21(6):541, 2019.
- [44] (77) Gregory Chaitin | Universidad de Buenos Aires - Academia.edu, Apr 2021. URL <https://uba.academia.edu/GregoryChaitin>. [Online; accessed 12. Apr. 2021].
- [45] Aurélien Géron. A Short Introduction to Entropy, Cross-Entropy and KL-Divergence, Feb 2018. URL <https://www.youtube.com/watch?v=ErfnhcEV108>. [Online; accessed 13. Apr. 2021].
- [46] 2 - How to Calculate a Correlation Matrix - Data Exploration for Machine Learning | Vertica, Nov 2020. URL <https://www.vertica.com/blog/in-database-machine-learning-2-calculate-a-correlation-matrix-a-data-explor> [Online; accessed 4. Feb. 2021].
- [47] ProClassify User's Guide - Cross-Validation Explained, Jun 2006. URL <https://genome.tugraz.at/proclassify/help/pages/XV.html>. [Online; accessed 1. Feb. 2021].
- [48] Donal Tobin. ETL Pipeline vs. Data Pipeline: What's the Difference? *Xplenty*, Jun 2020. URL <https://www.xplenty.com/blog/etl-pipeline-vs-data-pipeline>.
- [49] Data Engineering 101: How Do I Get Value From My Data - Seattle Data Guy, Mar 2020. URL <https://www.theseattledataguy.com/data-engineering-101/#page-content>. [Online; accessed 29. Oct. 2021].
- [50] What Exactly is a DAG?, Oct 2021. URL <https://www.astronomer.io/blog/what-exactly-is-a-dag>. [Online; accessed 29. Oct. 2021].
- [51] Info@numxl. com Spider Financial. MAPE - Error medio de porcentaje absoluto, Mar 2021. URL <https://support.numxl.com/hc/es/articles/215959443-MAPE-Error-medio-de-porcentaje-absoluto>. [Online; accessed 23. Mar. 2021].
- [52] Ashish Ahuja. Mean Absolute Scaled Error (MASE) in Forecasting - Ashish Ahuja - Medium. *Medium*, Jan 2021. URL <https://medium.com/@ashishdce/mean-absolute-scaled-error-mase-in-forecasting-8f3aec21968>.

- [53] Papers with Code - Papers With Code : Trends, Jul 2021. URL <https://paperswithcode.com/trends>. [Online; accessed 12. Jul. 2021].
- [54] Wikipedia. Keras - Wikipedia, la enciclopedia libre, Aug 2020. URL <https://es.wikipedia.org/w/index.php?title=Keras&oldid=128778152>. [Online; accessed 23. Mar. 2021].
- [55] Keras Team. Keras documentation: Callbacks API, Mar 2021. URL <https://keras.io/api/callbacks>. [Online; accessed 23. Mar. 2021].
- [56] Azure. DeepLearningForTimeSeriesForecasting, Mar 2021. URL [https://github.com/Azure/DeepLearningForTimeSeriesForecasting/blob/master/2\\_RNN.ipynb](https://github.com/Azure/DeepLearningForTimeSeriesForecasting/blob/master/2_RNN.ipynb). [Online; accessed 29. Mar. 2021].
- [57] Azure. DeepLearningForTimeSeriesForecasting, Mar 2021. URL [https://github.com/Azure/DeepLearningForTimeSeriesForecasting/blob/master/3\\_RNN\\_encoder\\_decoder.ipynb](https://github.com/Azure/DeepLearningForTimeSeriesForecasting/blob/master/3_RNN_encoder_decoder.ipynb). [Online; accessed 29. Mar. 2021].
- [58] Best Tools for Model Tuning and Hyperparameter Optimization - neptune.ai, May 2021. URL <https://neptune.ai/blog/best-tools-for-model-tuning-and-hyperparameter-optimization>. [Online; accessed 12. Jul. 2021].
- [59] Keras Tuner, Nov 2019. URL <https://keras-team.github.io/keras-tuner>. [Online; accessed 5. Apr. 2021].
- [60] CuPy, Apr 2021. URL <https://cupy.dev>. [Online; accessed 16. Apr. 2021].
- [61] CUDA Zone, Apr 2021. URL <https://developer.nvidia.com/cuda-zone>. [Online; accessed 16. Apr. 2021].
- [62] Rising Odegua. How to put machine learning models into production, Oct 2020. URL <https://stackoverflow.blog/2020/10/12/how-to-put-machine-learning-models-into-production>. [Online; accessed 2. Nov. 2021].
- [63] Aditya Chinchure. A production-grade Machine Learning API using Flask, Gunicorn, Nginx, and Docker — Part 1. *Medium*, May 2020. ISSN 4992-7238. URL <https://medium.com/technonerdz/a-production-grade-machine-learning-api-using-flask-gunicorn-nginx-and-dock>