UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADO EN INGENIERÍA INFORMÁTICA

# MODELADO Y SIMULACIÓN BASADA EN AGENTES DE LA GESTIÓN DE UN INVENTARIO DE PRODUCTOS PERECEDEROS USANDO APRENDIZAJE POR REFUERZO PROFUNDO BASADO EN MODELOS CALIBRADOS

# AGENT-BASED MODELING OF PERISHABLE INVENTORY MANAGEMENT USING CALIBRATED MODEL-BASED DEEP REINFORCEMENT LEARNING


Realizado por
CARLOS ALONSO OSORIO PENA

Tutorizado por
EDUARDO GUZMÁN DE LOS RISCOS
MARÍA VICTORIA BELMONTE MARTÍNEZ

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA



MÁLAGA, SEPTIEMBRE DE 2022

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA

# MODELADO Y SIMULACIÓN BASADA EN AGENTES DE LA GESTIÓN DE UN INVENTARIO DE PRODUCTOS PERECEDEROS USANDO APRENDIZAJE POR REFUERZO PROFUNDO BASADO EN MODELOS CALIBRADOS

# AGENT-BASED MODELING OF PERISHABLE INVENTORY MANAGEMENT USING CALIBRATED MODEL-BASED DEEP REINFORCEMENT LEARNING


Realizado por
**CARLOS ALONSO OSORIO PENA**

Tutorizado por
**EDUARDO GUZMÁN DE LOS RISCOS**
**MARÍA VICTORIA BELMONTE MARTÍNEZ**

Departamento
**LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN**

UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE DE 2022


Fecha de defensa: Octubre de 2022

# Resumen

Vivimos en un mundo en el que cerca de un 10 % de la población global sufre desnutrición mientras, al mismo tiempo, el 17 % de los alimentos producidos acaban en la basura. La catástrofe ecológica, social y médica que produce este desperdicio es devastadora, y por ello, desde este trabajo queremos aportar nuestro granito de arena para contribuir a paliar esta situación.

Parte de esa comida se desperdicia directamente en los supermercados, sin que acabe llegando al consumidor final, provocado por una gestión de inventario ineficiente. Este trabajo ha desarrollado un gestor de inventario de productos perecederos que sea capaz de encargar los pedidos para el día siguiente reduciendo lo máximo posible tanto la comida desperdiciada como las roturas de stock. Para ello se ha modelado un sistema basado en agentes apoyado por sistemas de aprendizaje por refuerzo profundo basado en modelos. Para minimizar el error de este sistema, se han calibrado las incertidumbres de la red neuronal bayesiana que utiliza, usando la técnica de calibración cuantil para regresión.

**Palabras clave:** gestión de inventario, modelado y simulación basada en agentes, aprendizaje por refuerzo profundo, calibración

# Abstract

We live in a world where near 10% of the global population suffers from malnutrition, while at the same time, 17% of the food that has been produced ends up in the garbage. This ecological, social and medical catastrophe produced by this waste is devastating. That is why we want to contribute with this project to mitigate this situation.
Part of this food ends up been wasted directly in the supermarkets, without even reaching the final consumer; this is the result of an inefficient inventory management. This project has developed a perishable inventory management utility that is able to place orders for the next day taking into account that it has to reduce as much as possible not only the food that can end up wasted, but also the stockouts. In order to do that, an agent-based modeling system using model-based deep reinforcement learning has been developed. In order to minimize the error, the uncertainties of the bayesian neural networks used by this system have been calibrated using quantile calibration for regression.

**Keywords:** inventory management, agent-based modeling, deep reinforcement learning, calibration

# Índice

# Lista de acrónimos

| | |
|---|---|
| **AA** | aprendizaje automático |
| **ARP** | aprendizaje por refuerzo profundo |
| **CPM** | control predictivo por modelo |
| **CRISP-DM** | *Cross Industry Standard Process for Data Mining* |
| **JIT** | *just-in-time* |
| **MC** | Monte Carlo |
| **MSBA** | modelado y simulación basada en agentes |
| **PDM** | proceso de decisión de Markov |
| **RNA** | red neuronal artificial |
| **RNB** | red neuronal bayesiana |
| **RNP** | red neuronal prealimentada |
| **RNR** | red neuronal recurrente |
| **SMA** | sistema multiagente |

# 1

# Introducción

## 1.1. Motivación

Para que un alimento llegue al consumidor final, se utilizan una gran cantidad de recursos tanto físicos como humanos: el cultivo inicial de las materias primas, los procesos de transformación (si son necesarios), el envasado, transporte y posterior venta al por mayor y/o al público general.

Según el Índice de desperdicios de alimentos (United Nations Environment Programme, 2021), en el año 2019 el 17 % de la producción mundial de alimentos acabó en la basura. Esto tiene unos efectos devastadores, tanto a nivel económico como a nivel medioambiental.

Parte de este desperdicio de alimentos se da en los propios supermercados, incluso antes de que lleguen al consumidor final.

En los supermercados existe la figura de los gestores de inventarios, que se encargan de predecir qué cantidades de cada producto son necesarias en el futuro próximo, para no encontrarnos en situaciones de exceso o escasez y, por tanto, para una gestión eficiente del estocaje.

Cuando se trata de alimentos perecederos, es aún más importante dar una aproximación lo más exacta posible. La escasez va a suponer una rotura de stock, con el consiguiente agravio para la cadena. Y el exceso va a producir un desperdicio de productos debido a su deterioro o caducidad.

Este trabajo está motivado por la necesidad de paliar el desperdicio de alimentos. Para ello, se pretende implementar un gestor de inventario de productos perecederos lo más eficiente posible.

## 1.2. Objetivos

El principal objetivo de este trabajo es desarrollar y documentar un gestor de inventario de productos perecederos.

Para cumplir el objetivo, se combinará el modelado y simulación basada en agentes (MSBA), junto con técnicas de aprendizaje por refuerzo profundo basado en modelos calibrados inspiradas en el trabajo *Calibrated Model-Based Deep Reinforcement Learning* (Malik y cols., 2019). Como inventario, se utilizarán los datos de Corporación Favorita, que se pueden encontrar en el repositorio Kaggle (Corporación Favorita, 2018).

El software que se desarrollará tendrá como entrada un conjunto de los datos de Corporación Favorita, y como salida, la predicción hecha y la diferencia con los valores reales (número y porcentaje de unidades desperdiciadas y de unidades sin stock).

Además, se realizará una comparativa entre los valores obtenidos por el software y los que obtuvo el estudio anteriormente citado.

## 1.3. Tecnologías a utilizar

Estas son las tecnologías con las que se va a trabajar a la hora de desarrollar el proyecto:

- **Python**

  El trabajo se desarrollará fundamentalmente en el lenguaje de programación Python.

  Se trata de un lenguaje muy utilizado a día de hoy para el desarrollo de aplicaciones de ciencia de datos, y además posee un framework para el MSBA.

- **Mesa**

  Para el modelado y simulación de agentes se utilizará el framework Mesa (Kazil, Masad, y Crooks, 2020).

  Permite al desarrollador implementar el modelo basado en agentes en Python, visualizar los resultados en un navegador web y analizarlos con las herramientas de análisis de datos de Python.

- **R**

  Para el análisis del conjunto de datos Corporación Favorita, se utilizará el lenguaje de programación R.

Se ha preferido usar este lenguaje de programación por el hecho de ser de código libre (al contrario de otros lenguajes, como Matlab), tener un gran número de bibliotecas que aumentan sus posibilidades de uso, y ser uno de los lenguajes de programación más utilizados para el análisis de datos.

## 1.4. Metodología de trabajo

En este trabajo se ha utilizado la metodología *Cross Industry Standard Process for Data Mining* (CRISP-DM) (Wirth y Hipp, 2000), que es la metodología más utilizada para trabajos relacionados con la ciencia de datos.

Las fases de esta metodología no son estrictas, lo que permite avanzar o retroceder en el ciclo de fases según la situación lo requiera y se asemeja, por tanto, a los desarrollos ágiles.

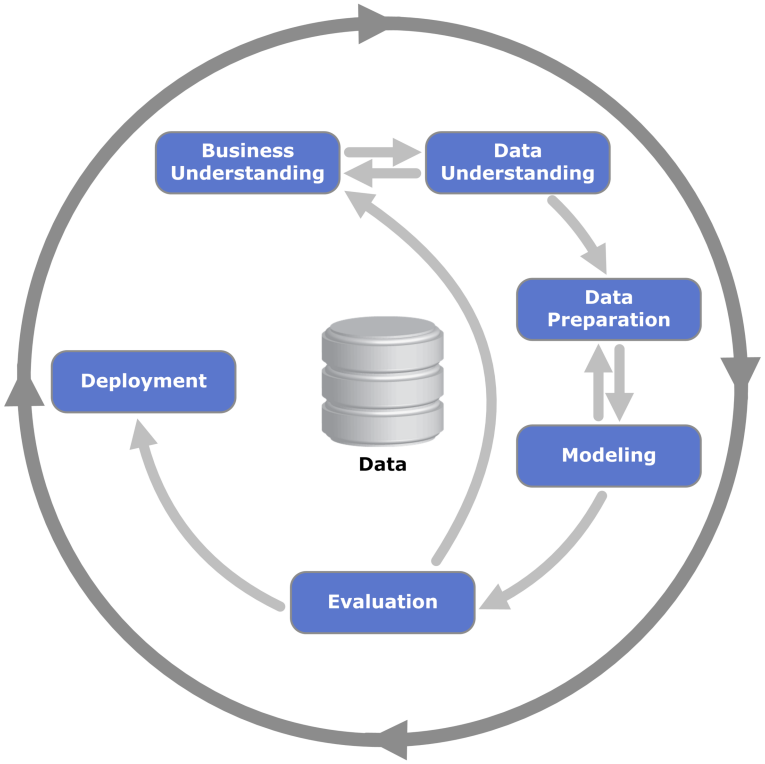


Figura 1: Diagrama de procesos de CRISP-DM (Jensen, 2012)

En la figura 1 podemos ver las distintas fases de la metodología CRISP-DM, que pasamos a describir a continuación:

- **Comprensión del negocio**

  Fase inicial en la que se trata de entender los principales objetivos del proyecto y enlazarlos con la ciencia de datos.

- **Comprensión de los datos**

  Estudiar detenidamente el conjunto de datos sobre el que se va a trabajar.

- **Preparación de los datos**

  Hacer las manipulaciones necesarias al conjunto de datos para acabar con un conjunto final sobre el que trabajar.

- **Modelado**

  Se trata de construir un modelo para nuestro proyecto. Para ello, se seleccionarán, calibrarán y aplicarán los distintos algoritmos de modelado para conseguir sus valores óptimos.

- **Evaluación**

  En esta fase se evalúan y se revisan todos los procesos que se han hecho hasta el momento para estar seguros de que se cumplen los objetivos.

- **Implantación**

  Fase final en la que se realiza el despliegue del proyecto, así como su informe final.

## 1.5. Estructura de la memoria

Se ha decidido estructurar la memoria dividiendo el contenido en los siguientes capítulos:

- **Introducción**

  Como su nombre indica, se trata de un capítulo introductorio, donde se dan las motivaciones que han dado lugar a la realización de este trabajo, se indican cuáles son los objetivos que se pretenden alcanzar, y se explican las tecnologías y la metodología a seguir. De esta forma, el lector tiene una noción general de lo que se va a desarrollar a continuación y por qué se ha decidido hacerlo.

- **Estado del arte**

  En este capítulo se da una visión general de las publicaciones e investigaciones más relevantes que se han desarrollado en el tema fundamental sobre el que trata el trabajo. Más concretamente se habla del estado actual del modelado basado en agentes, del aprendizaje automático, de la minería de agentes (la combinación de ambas técnicas) y de la teoría de inventarios.

- **Fases del proyecto**

  Se desarrollan y documentan todas las fases y tareas que conforman la metodología de trabajo elegida y que, una vez aplicadas, dan lugar a la consecución de los objetivos proyecto.

- **Estudio de resultados**

  Se muestran los resultados obtenidos en el trabajo, se estudia la importancia de los mismos, y se comparan con otros trabajos hechos sobre el mismo conjunto de datos.

- **Conclusiones y líneas futuras**

  En este último capítulo se habla de las conclusiones que se han sacado tras la realización del trabajo; de lo que este ha aportado; y se da una breve visión de qué caminos podría tomar alguien que quisiera utilizar este trabajo como base para proyectos futuros.

# 2

# Estado del arte

## 2.1. Aprendizaje automático

"*Se dice que un programa informático aprende de una experiencia $E$ con respecto a una clase de tareas $T$ y una medida de rendimiento $P$, si su rendimiento al realizar $T$, medido por $P$, mejora con la experiencia $E$.*" (Mitchell, 1997).

El aprendizaje automático (AA) es una rama de la inteligencia artificial dedicada al uso de datos y algoritmos para imitar el modo en que las personas aprenden, mejorando su precisión gradualmente.

En la actualidad, el AA se considera una herramienta muy útil para un amplio rango de investigaciones científicas y de aplicaciones industriales. Su mayor fuerte es el de descubrir patrones complejos y examinar relaciones no lineales.

### 2.1.1. Categorías

El AA se divide en tres categorías amplias, como se puede ver en la figura 2, aunque hay algoritmos que no encajan del todo en una categoría o utilizan métodos de varias categorías.

- **Aprendizaje supervisado**

  En el aprendizaje supervisado (Heidenreich, 2018), el algoritmo genera un modelo a través del entrenamiento con un conjunto de datos etiquetados, que después utilizará para predecir el resultado con nuevos datos.

  Dependiendo de si la salida es continua o discreta, hablaremos de un problema de regresión o un problema de clasificación respectivamente.

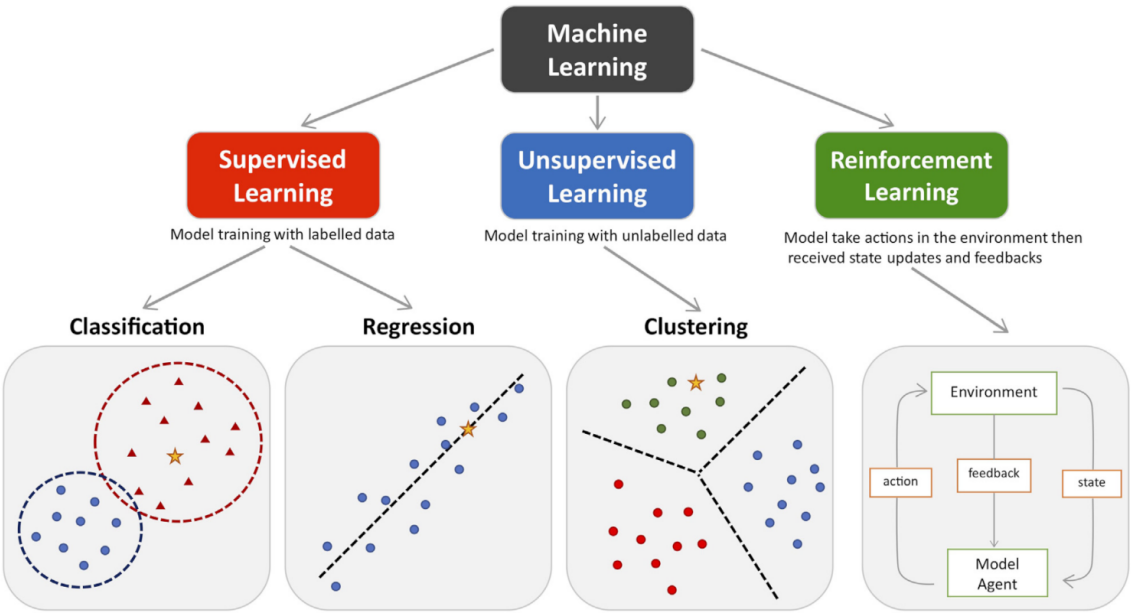


Figura 2: Categorías principales del aprendizaje automático (Peng y cols., 2021)

En este tipo de aprendizaje, por tanto, será necesario tener un conjunto de datos etiquetados (que tengan la salida deseada). En primer lugar, se dividirá el conjunto de datos en entrenamiento y prueba. El conjunto de entrenamiento será utilizado para generar el modelo. El conjunto de prueba se usará para hacer predicciones y comprobar la eficacia del algoritmo.

La mayoría de algoritmos tienen una serie de parámetros que se pueden modificar para encontrar la eficacia adecuada.

En este trabajo implementaremos en primer lugar un algoritmo de aprendizaje supervisado, más concretamente *Random Forest*. Se ha elegido este algoritmo en particular por su sencillez (Thorn, 2020) y por haber trabajado con él en otros proyectos. Al utilizar un algoritmo de aprendizaje supervisado, hacemos que nuestro sistema esté listo para que pueda trabajar con cualquier otro algoritmo de ese tipo (ver la *guía de programación*).

- **Aprendizaje no supervisado**

  El aprendizaje no supervisado (Heidenreich, 2018) se utiliza para encontrar patrones ocultos en conjuntos de datos no etiquetados. Este tipo de aprendizaje no necesita que el conjunto de datos esté etiquetado (contenga la salida esperada) e incluso puede estar incompleto o contener datos pocos relevantes.

  La capacidad de agrupar y de descubrir similitudes y diferencias en el conjunto de datos lo hace muy útil para análisis de exploración de datos, estrategias de ventas

cruzadas, segmentación de clientes, reconocimiento de imágenes y patrones, etc.

Se distinguen dos tipos fundamentales de aprendizaje no supervisado:

- *Clustering:* Se trata de dividir los datos en grupos (o clústeres), atendiendo a las similitudes o diferencias que el modelo encuentre. Se utilizan fundamentalmente para clasificar grupos de datos no preprocesados.

- *Asociación:* Se trata de identificar relaciones entre los parámetros de un conjunto de datos usando técnicas basadas en reglas. Esta técnica se utiliza, por ejemplo, para el análisis de la cesta de la compra (encontrar la relación entre la venta de un producto con la venta de otros productos dependiendo del comportamiento del cliente).

- **Aprendizaje por refuerzo**

  En el caso de aprendizaje por refuerzo (Heidenreich, 2018), el algoritmo aprende a tomar la mejor acción en el entorno a través de un sistema de recompensas utilizando el método de ensayo y error.

  Esta categoría, al contrario que las anteriores, no está basada en si el conjunto de datos está etiquetado o no, sino que está basada en el comportamiento del sistema. El algoritmo va eligiendo acciones en el entorno y recibiendo una respuesta positiva o negativa; con el tiempo, comienza a diferenciar entre las acciones buenas y malas y a elegir la correcta, consiguiendo cada vez una mayor precisión.

  Este problema se suele modelar como un proceso de decisión de Markov (PDM) (Sutton y Barto, 2017) $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, donde $\mathcal{S}$ es el conjunto de estados posibles (también llamado espacio de estados), $\mathcal{A}$ es el conjunto de acciones (también llamado espacio de acciones), $\mathcal{P}$ es la función de transición (o la probabilidad de pasar al siguiente estado, dados una acción y un estado actual), y $\mathcal{R}$ es la recompensa esperada.

  Tenemos un agente (nuestro algoritmo de aprendizaje por refuerzo) que interactúa con su entorno en una secuencia discreta de intervalos temporales $t = 0, 1, 2, 3....$ En cada intervalo $t$, el agente se encuentra en un estado $s_t \in \mathcal{S}$, y realiza una acción $a_t \in \mathcal{A}$ siguiendo una política $\pi(a_t|s_t)$ (que describe el comportamiento del agente), donde recibe una recompensa $r_t \in \mathcal{R} \subset \mathbb{R}$ y pasa al estado $s_{t+1}$ de acuerdo con la probabilidad de transición $\mathcal{P}(s_{t+1}|s_t, a_t)$.

  El objetivo del agente es el de aprender una política $\pi : \mathcal{A} \times \mathcal{S}$ que maximice la recompensa acumulada.

Podemos encontrar dos tipos fundamentales de algoritmos de aprendizaje por refuerzo (Sutton y Barto, 2017):

- ***Sin modelos***: Se trata de métodos más simples (y por ende, más asequibles computacionalmente) que no utilizan ni planificación ni modelos, ni evalúan cómo su entorno cambia en respuesta a una acción, y que por tanto su aprendizaje se basa explícitamente en el paradigma de ensayo-error. La política se aprende a través de la experiencia.

  Este tipo de algoritmos son muy útiles cuando la mayor dificultad del problema reside en construir un modelo del entorno lo suficientemente preciso.

- ***Basado en modelos***: Los algoritmos basados en modelos intentan aprender un modelo del entorno a través de la interacción con el mismo, y el aprendizaje del modelo se suele llevar a cabo mediante aprendizaje supervisado. La política se aprende a través del modelo.

  Los algoritmos basados en modelos tienen la ventaja de necesitar un conjunto de datos menor que los sin modelo para aprender una política; además, al aprender un modelo, este se puede utilizar para diferentes tareas (y por tanto aprender diferentes políticas), frente a los sin modelo, que solo aprenden una política. Por contra, como mayor desventaja nos encontramos con que este tipo de algoritmos tienen dos fuentes de incertidumbre: el modelo y la política; frente a una única fuente de incertidumbre (la política) de los algoritmos sin modelo.

  Las estimaciones de las incertidumbres del modelo pueden ser erróneas, haciendo que el agente sobrestime su confianza y pase por alto algunas situaciones peligrosas. Una solución a este problema es la **calibración** de los modelos (Malik y cols., 2019) para mejorar la eficacia del algoritmo con un coste computacional mínimo.

  Se entiende que un modelo de transición $\widehat{T}(s_{t+1}|s_t, a_t)$ está calibrado si cuando asigna una probabilidad de 0,7, por ejemplo, a un evento (como una transición de estado $(s_t, a_t, s_{t+1})$), esa transición debería ocurrir el 70 % de las veces.

  La mayor ventaja de los algoritmos basados en modelos calibrados es que nos permiten hacer una planificación más eficaz usando algoritmos estándares (Malik y cols., 2019).

Una de las mayores dificultades del aprendizaje por refuerzo es el problema de exploración vs explotación. O en otras palabras, cuándo debe el agente utilizar

acciones no óptimas para poder explorar el entorno y construir un mejor modelo, y cuándo debe explotar las acciones óptimas para progresar adecuadamente.

### 2.1.2. Redes neuronales artificiales

Una red neuronal artificial (RNA) (Kavlakoglu, 2020) utiliza un conjunto de algoritmos basados en cómo funciona el cerebro humano.

Las redes neuronales biológicas funcionan en un sistema jerárquico por niveles, donde la información pasa de un nivel a otro, haciendo un procesado en cada etapa. Como se ve en la figura 3, las RNA replican dicho comportamiento. Podemos dividir las capas de las RNA en tres: la capa de entrada, donde se recibe la información del entorno; las capas ocultas, que procesan la información; y la capa de salida, que da el resultado.

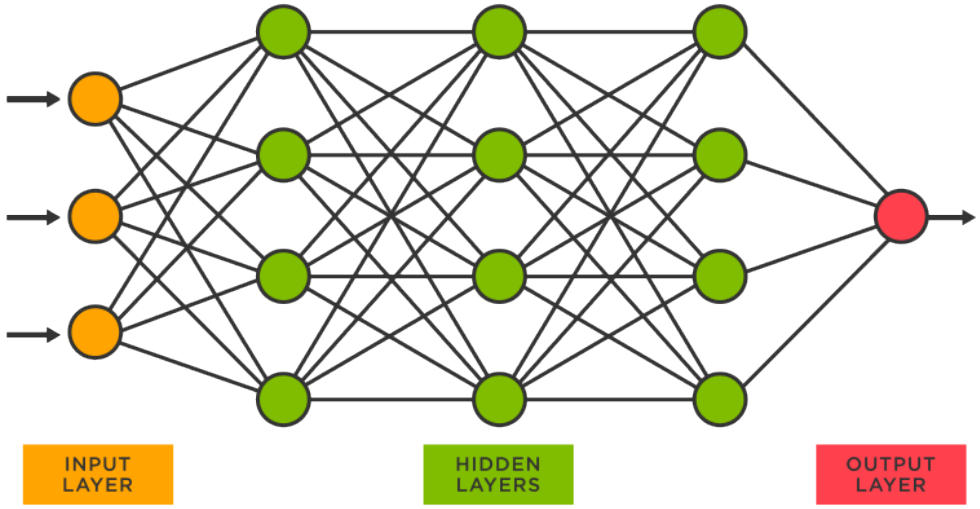


Figura 3: Diagrama de una red neuronal (TIBCO Software Inc., 2021)

En una RNA básica, una neurona consta de entradas ($x$), pesos ($w$), un umbral y una salida, y se puede expresar matemáticamente como:

$$\sum_{i=1}^{n} w_i x_i - umbral$$

La salida se puede calcular a través de una función de activación, como puede ser la siguiente:

$$f(x) = \text{máx}(0, x)$$

Esta función de activación llamada ReLU (activación lineal rectificada) (Agarap, 2018) es bastante usada en RNA por su eficiencia (simplicidad computacional) y por su dispersión (al convertir muchos valores a cero). La función devuelve cero si el valor es negativo (y por tanto no hay activación), y el propio valor si es positivo.

Un tipo especial de RNA es la red neuronal bayesiana (RNB) (Kononenko, 1989). En este tipo de redes neuronales, tanto los pesos como las salidas son modelados como distribuciones de probabilidad en lugar de como valores deterministas, y por ende, nos permiten medir la incertidumbre del sistema.

Una forma de obtener estas distribuciones de probabilidad en una RNA es a través del método *Monte Carlo (MC) dropout* (Gal y Ghahramani, 2015). Este método usa las capas de abandono (*dropout layers*) en las redes neuronales para obtener modelos probabilísticos. Este tipo de capas ignoran aleatoriamente un porcentaje de las neuronas de la red. Al usar estas capas tanto en la fase de entrenamiento como en la de prueba, las salidas de la red neuronal dejan de ser deterministas, y se puede conseguir con múltiples ejecuciones un número de muestras a partir de las cuales obtener la distribución de probabilidad de la red.

El empleo de las redes neuronales dentro del aprendizaje automático se da en los siguientes campos:

- **Aprendizaje profundo**

  Cuando una RNA tiene más de una capa oculta, podemos hablar de una red neuronal profunda. Este tipo de redes neuronales son la base del aprendizaje profundo (IBM Cloud Education, 2020), que es un tipo de AA que combina las redes neuronales con el aprendizaje automático.

  El aprendizaje profundo es utilizado en campos muy diversos como el análisis de imágenes, reconocimiento de voz, procesado del lenguaje natural, sistemas de recomendación, vehículos autónomos, etc.

- **Aprendizaje por refuerzo profundo**

  La combinación del aprendizaje profundo con el aprendizaje por refuerzo da lugar al aprendizaje por refuerzo profundo (ARP) (Li, 2017). En este tipo de algoritmos, o bien la política óptima o bien la función de transición se modelan a través de redes neuronales.

  Uno de los mayores usos del ARP es el de conseguir implementar las técnicas de aprendizaje por refuerzo en problemas de alta dimensionalidad que no eran factibles de resolver sin utilizar las RNA.

  El ARP se divide también en las mismas categorías que el aprendizaje por refuerzo: sin modelos y basado en modelos (que pueden estar o no calibrados).

En este trabajo haremos uso en segundo lugar de un sistema de ARP basado en modelos calibrados, para una estimación más compleja y más eficiente del problema de gestión de

inventario.

## 2.2. Modelado y simulación basada en agentes

El MSBA es una técnica de modelado de abajo a arriba en la que podemos entender fenómenos a gran escala a través de la simulación de comportamientos a pequeña escala, como son las acciones e interacciones entre agentes y un entorno.

El MSBA se usa para comprender el comportamiento de sistemas complejos.

### 2.2.1. Definición de agente

Es difícil encontrar una definición exacta de lo que es un agente, dado que no hay consenso sobre cómo definirlo. En *Is it an Agent, or just a Program?* (Franklin y Graesser, 1997) se dan hasta 10 definiciones distintas de agentes, que varían desde las más básicas a las más restrictivas, probablemente según los ejemplos de agentes que el autor tenía en su cabeza en el momento de formular la definición. En la mayoría de ellas, se describe al agente como un componente software (o hardware) que interactúa con un entorno y con otros agentes para conseguir un objetivo.

Además, en *Agent Theories, Architectures, and Languages: A Survey* (Wooldridge y Jennings, 1995) se detallan las características que los agentes deben tener:

- *Autonomía*: Operar sin intervención humana y tener control sobre sus acciones y sus estados.
- *Habilidad social*: Interactuar con otros agentes.
- *Reactividad*: Percibir el entorno y responder a los cambios que ocurren en él.
- *Proactividad*: Ser capaces de exhibir un comportamiento enfocado a conseguir un objetivo y tomar la iniciativa.

### 2.2.2. Arquitectura de agentes

Wooldrige describe la arquitectura de agentes como la arquitectura software cuya misión es la de respaldar el sistema de decisión de un agente (Wooldridge, 2001). Las arquitecturas de agentes son, por tanto, la base para crear agentes de forma parecida a como se crea un objeto en una clase (Anthony, Soon, On, Alfred, y Lukose, 2014).

Como categorías amplias de arquitecturas de agentes podemos encontrar los tres siguientes tipos:

- **Deliberativa**

  Una de las primeras arquitecturas de agentes en surgir. En esta arquitectura, el agente guarda una representación simbólica del mundo y toma sus decisiones siguiendo un razonamiento simbólico. Los procesos internos de un agente deliberativo pueden ser bastante complejos.

  Un ejemplo de modelo muy usado para este tipo de arquitecturas es el *Believe-Desire-Intention*, donde se modelan los agentes en base a sus creencias (información sobre el estado del agente y del entorno), deseos (objetivos) e intenciones (planes).

- **Reactiva**

  Hay muchos problemas que no pueden ser resueltos por arquitecturas deliberativas. Esto llevó a que algunos investigadores se cuestionaran la viabilidad de dicha arquitectura y desarrollaran la arquitectura reactiva.

  En esta arquitectura, el agente basa sus decisiones fundamentalmente en la situación actual y no necesita una representación simbólica del mundo.

  Los sistemas basados en esta arquitectura son bastante simples. Estos sistemas tienen una serie de reglas relativamente simples ordenadas por prioridad (unas tienen precedencia sobre otras), y se va eligiendo la regla adecuada según la información que se recibe del entorno y la prioridad entre las reglas.

- **Híbrida**

  Muchos investigadores llegaron a la conclusión de que ninguna de las dos arquitecturas anteriores es adecuada para resolver la mayoría de problemas. De ahí que surgiera una arquitectura híbrida que usa los principios de ambas.

  La idea es la de unir dos agentes (subsistemas) en uno. Uno de los agentes sería el deliberativo, con una representación simbólica del mundo, que toma las decisiones siguiendo la lógica simbólica. El otro sería un agente reactivo, capaz de actuar según el entorno sin un razonamiento complejo.

  Al tener varios subsistemas, hace falta alguna forma de saber qué decisión tomar. Para ello, estos subsistemas se ordenan en jerarquías por capas, donde cada capa tiene un nivel más alto de abstracción.

### 2.2.3. Sistemas multiagente

Un sistema multiagente (SMA) es un sistema formado por múltiples agentes que interactúan y se comunican entre ellos para conseguir un objetivo.

La interacción entre los distintos agentes puede ser cooperativa o competitiva. En las interacciones cooperativas, los agentes colaboran entre sí para llegar a un objetivo común; la idea en este caso es que los agentes se distribuyan y compartan los conocimientos y capacidades para resolver problemas. En las interacciones competitivas, por contra, los agentes compiten entre ellos para alcanzar cada uno sus objetivos individuales.

Algunos de los factores que caracterizan los SMA son el entorno, los agentes y las interacciones entre agentes (Huhns y Singh, 1998). Además, también se caracterizan por el tamaño de los equipos, la reconfigurabilidad, la composición y la topología de comunicación (Dudek, Jenkin, Milios, y Wilkes, 1993).

Los estándares que regulan los SMA vienen dados fundamentalmente por FIPA (*Foundation for Intelligent Physical Agents*), un comité de estándares IEEE creado en 2005 sobre la antigua organización suiza del mismo nombre fundada en 1996.

En particular, la plataforma de agentes FIPA (Poslad, Buckle, y Hadingham, 2000) define la arquitectura que deben seguir los SMA, como se ve en la figura 4.

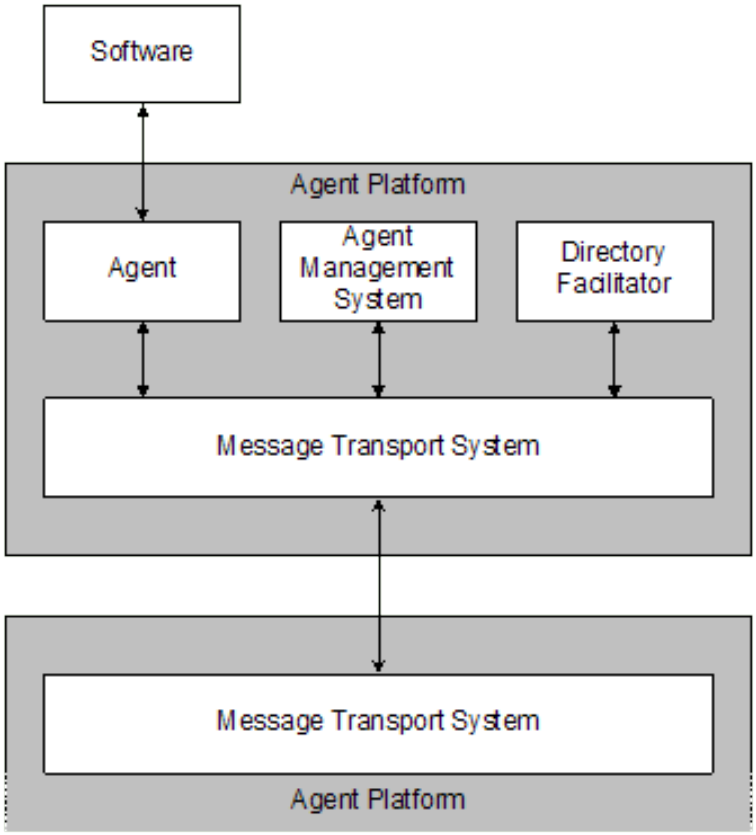


Figura 4: Plataforma de agentes FIPA (Foundation for Intelligent Physical Agents, 2002)

Los principales componentes en esta arquitectura son:

– *Agente:* Son la parte fundamental de la plataforma. Cada agente encapsula uno o más servicios dentro de un modelo de ejecución.

– *Sistema de gestión de agentes (AMS):* Se encarga de ofrecer servicios de administración del agente y de gestión de su ciclo de vida, manteniendo un directorio de identificadores de agente y del estado de ejecución de cada agente.

– *Facilitador de directorio (DF):* Se trata de un componente opcional. El facilitador de directorio permite a los agentes registrar sus servicios y pedir información sobre los servicios disponibles dentro de la comunidad de agentes.

– *Sistema de transporte de mensajes (MTS):* Su función es la de proveer de un servicio de entrega y recepción de mensajes tanto entre los distintos agentes como con servicios externos de otras plataformas.

### 2.2.4. Minería de agentes

El término "minería de agentes" se acuña en *Introduction to Agent Mining Interaction and Integration* (Cao, 2009) para representar la integración entre campos como MSBA y AA (o también minería de datos, de ahí el nombre).

La minería de agentes es un campo que cada vez se está extendiendo más, y sobre el que se están escribiendo guías y artículos sobre cómo conectar ambas técnicas y las ventajas que ofrece. En la figura 5 se muestran las cuatro principales opciones para integrar el aprendizaje automático en MSBA:

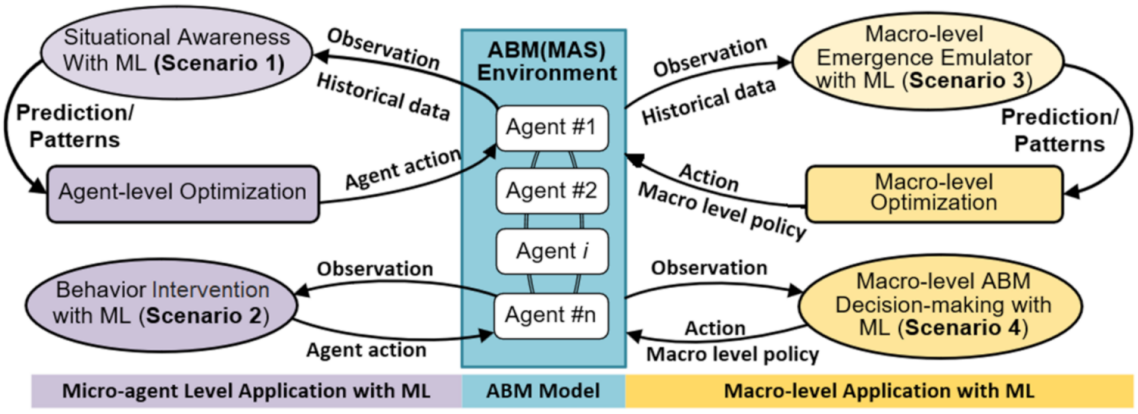


Figura 5: Ciclo de un modelo basado en agentes usando aprendizaje automático a nivel de micro-agente (izquierda) o a macro-nivel (derecha) (Zhang y cols., 2021)

– *Aprendizaje situacional a nivel micro-agente:* Se corresponde al escenario 1 en la figura.

  El agente utilizará el AA para predecir variables o patrones, y a través de estas predicciones, elegir el comportamiento óptimo.

– *Intervención de comportamiento a nivel micro-agente:* Se corresponde al escenario 2 en la figura.

  En este caso, el agente además de predecir variables o patrones, utilizará el AA para reforzar su comportamiento (sus tomas de decisión).

– *Afloramiento a macro-nivel:* Se corresponde al escenario 3 en la figura.

  Aquí nos encontramos con que el algoritmo de AA es usado por el sistema de control del MSBA para extraer o generar algún patrón o variable a través del comportamiento del conjunto de agentes, y tomar así decisiones a macro-nivel.

– *Toma de decisiones a macro-nivel:* Se corresponde al escenario 4 en la figura.

  En este escenario, el agente además de usar el algoritmo de aprendizaje profundo para el afloramiento de patrones o variables, lo usará también para reforzar su comportamiento.

En este trabajo utilizaremos la minería de agentes siguiendo las pautas del escenario 1, donde el algoritmo de aprendizaje automático se utilizará a nivel de micro-agente para predecir variables (concretamente el número de ventas por artículo y tienda para los días siguientes).

## 2.3. Teoría de inventarios

Mantener un inventario es una práctica muy común en la mayoría de negocios. De esta forma, se tienen almacenados una serie de productos para satisfacer la demanda de los futuros clientes. La teoría de inventarios trata de minimizar el coste que les supone a las compañías la gestión de inventarios.

### 2.3.1. Factores

Los principales factores que influyen en el rendimiento económico de la gestión de inventario, tal y como se describen en *Inventory Theory* (Zappone, 2006), y que van a formar parte de cualquier modelo de gestión de inventario, son:

– *Coste de adquisición:* El coste de pedir o manufacturar el producto.

– *Coste de almacenamiento:* Incluye todos los costes relacionados con el almacenamiento, como pueden ser el alquiler del local, el seguro, la compañía de seguridad, etc.

- *Coste de ruptura:* Aquí se pueden diferenciar dos tipos:
  - Con retraso, donde el coste en el que se incurre es el de la confianza del cliente, que puede no querer volver a comprar en el negocio después de la experiencia.
  - Sin retraso, donde, o bien hay que hacer un pedido exprés con el coste que supone, o bien se pierde esa venta. En el segundo caso, el coste sería el de perder la venta, más el coste de pérdida de confianza del cliente.
- *Coste de rescate:* Se trata del valor de los productos que ya no se quieren en inventario y el negocio quiere deshacerse de ellos poniéndolos en rebaja.
- *Coste de oportunidad:* Se refiere a lo que se ha dejado de ganar por tener el dinero atado a la gestión de almacén en lugar de tenerlo invertido.

### 2.3.2. Modelos

En los modelos de la teoría de inventarios, se utilizan los factores anteriores para realizar estimaciones sobre el reabastecimiento de stock en base a ciertas hipótesis. Destacamos dos:

- **Modelos determinísticos**

  En los modelos determinísticos se asume que todos los factores asociados a la gestión de inventario son conocidos, y que por tanto no hay ningún tipo de incertidumbre asociada con la demanda y el reabastecimiento de stock.

  Uno de los modelos más conocidos es EOQ (*Economic Ordering Quantity*), desarrollado por Ford Harris en 1913, que presupone que la demanda de un producto es constante dentro del año y que el reabastecimiento se produce sin retrasos.

- **Modelos estocásticos**

  En la mayoría de ocasiones, la demanda no es un valor constante y conocido, lo que nos lleva a los modelos estocásticos. Estos modelos utilizan distribuciones de probabilidad donde la demanda, y posiblemente otros factores, son variables aleatorias.

  Podemos encontrar dos variantes fundamentales: único periodo y múltiples periodos. En la variante de único periodo, el reabastecimiento se calcula solo para el momento actual, dado que el artículo pierde su valor pasado un periodo; esta variante se usa para artículos perecederos u otros artículos como periódicos o revistas. En la variante de múltiples periodos, el reabastecimiento se calcula teniendo en cuenta periodos futuros, dado que el artículo no pierde su valor.

En la metodología *just-in-time* (JIT) (Jenkins, 2020), se busca que el abastecimiento se produzca justo cuando el producto se necesita (es decir, cuando va a ser adquirido por el cliente), intentando incrementar la eficiencia y reducir el desperdicio. Este sistema requiere realizar una estimación muy precisa, además de mantener una relación muy dinámica con proveedores de confianza.

En este trabajo se ha utilizado un modelo estocástico de único periodo usando la metodología JIT, realizando la estimación de la demanda mediante el aprendizaje automático, y simplificando la mayoría de factores asociados a la teoría de inventarios (asumiendo coste cero).

# 3

# Fases del proyecto

Para seguir de manera correcta las fases de la metodología CRISP-DM, nos hemos guiado por las directrices del manual de IBM (IBM Corporation, 2011).

## 3.1. Comprensión del negocio

En esta fase del proyecto debemos comprender qué objetivos queremos alcanzar desde un punto de vista de negocio, y enlazar esos objetivos con la ciencia de datos.

### 3.1.1. Objetivos de negocio

Dada la ineficacia de algunos gestores de inventarios en las cadenas de suministro de los supermercados, se pretende desarrollar un gestor eficiente. Para ello, se analizará un conjunto de datos históricos de ventas y se aplicarán técnicas de aprendizaje automático.

El objetivo de negocio es el disminuir el número de roturas de stock y de alimentos desperdiciados en un supermercado.

El estudio se considerará como satisfactorio si se consigue que el gestor sea igual o mejor al que consiguieron en el estudio *Calibrated Model-Based Deep Reinforcement Learning*, es decir, capaz de obtener un porcentaje de roturas de stock del 2,8 % o inferior, y de desperdicio de comida por debajo del 2,2 % asumiendo, al igual que en dicho estudio, que la caducidad de los productos es como mucho de 5 días.

### 3.1.2. Evaluación de la situación

Al tratarse de un proyecto académico individual, será el alumno el que realice todas las tareas de ciencia de datos, mientras que los tutores tendrán el rol de supervisores del

proyecto y hablarán en nombre del cliente.

En cuanto a los datos disponibles, se va a trabajar con los de Corporación Favorita, que tiene todas las operaciones de una cadena de supermercados ecuatoriana desde 2013-01-01 a 2017-08-15. Es una base de datos bastante extensa y con la que deberíamos poder trabajar sin más riesgo.

Los dos riesgos a contemplar en este proyecto son el riesgo temporal y el riesgo de resultados.

Por un lado, se trata de un proyecto acotado en el tiempo, y se puede dar la circunstancia de que la carga de trabajo sea mayor de lo planificado y por tanto no se acabe a tiempo.

Por otro lado, nos podríamos encontrar con el riesgo de que el modelo que se diseñe no cumpla con las expectativas y los objetivos marcados.

### 3.1.3. Objetivos de ciencia de datos

Una vez que tenemos claros los objetivos de negocio, toca trasladarlos a objetivos de ciencia de datos.

Nos encontramos ante un problema de predicción. Para poder resolverlo, utilizaremos un sistema basado en agentes, que a su vez usen técnicas de ARP basado en modelos calibrados.

Como nos vamos a basar para las técnicas de ARP en el estudio *Calibrated Model-Based Deep Reinforcement Learning*, en principio utilizaremos valores similares a los de ellos en el conjunto de datos: nos quedaremos con los 100 alimentos más vendidos (cada agente se encargará de uno de los alimentos), como conjunto de entrenamiento para nuestro sistema utilizaremos todas las transacciones de 2014-01-01 a 2016-05-31, y como conjunto de prueba, de 2016-06-01 a 2016-08-31.

Como criterio de éxito, utilizaremos uno similar al de negocio: Consideraremos el estudio como satisfactorio si en el conjunto de prueba se predicen correctamente las cantidades a reponer de cada producto, con el mismo margen de error que en los objetivos de negocio: entre el +2,2 % y -2,8 % (es decir, que no pida más de un 2,2 % ni menos de un 2,8 % de las cantidades vendidas).

### 3.1.4. Plan de proyecto

En el cuadro 1 mostramos el plan de proyecto con las tareas de cada fase que faltan por hacer, junto a los riesgos asociados a las mismas.

| Fase | Tarea | Tiempo | Riesgos |
|---|---|---|---|
| Comprensión datos | Análisis del conjunto de datos | 25 horas | Problemas con los datos |
| | Descripción de los datos | 15 horas | |
| Preparación datos | Preparación | 15 horas | Problemas con los datos |
| Modelado | Selección de algoritmos | 1 hora | No encontrar los algoritmos adecuados |
| | Construcción del modelo | 55 horas | No encontrar el modelo adecuado |
| | Calibración de parámetros | 15 horas | No encontrar parámetros válidos y darnos cuenta de que el modelo no es adecuado |
| | Comprobación y documentación | 30 horas | Encontrar fallos que hagan replantear el modelo |
| Evaluación | Revisar el proceso | 10 horas | No lograr objetivos y tener que empezar de nuevo |
| | Determinar siguientes pasos y documentar | 15 horas | |
| Implantación | Plan de implantación | 25 horas | Problemas técnicos con la implantación |
| | Informe final | 25 horas | |
| | Revisión de todo el proceso | 10 horas | Encontrar fallos graves en la revisión final |

Cuadro 1: Plan de proyecto

## 3.2. Comprensión de los datos

En esta fase se estudiará con detenimiento el conjunto de datos sobre el que se pretende trabajar.

### 3.2.1. Recopilación inicial

En este proyecto vamos a trabajar con datos ya existentes. En concreto, se va a trabajar con la base de datos proporcionada por Corporación Favorita, que recoge las transacciones de una cadena de supermercados ecuatoriana.

### 3.2.2. Descripción

El conjunto de datos que utilizaremos se compone de los siguientes 6 ficheros CSV que pasamos a describir:

– *Holiday events:* Todas las fechas que han sido festivos en Ecuador (o parte del país).

Consta de 350 filas.

Tenemos estos campos: `date` (*fecha*): la fecha de la festividad. `type` (*cadena*): el tipo de fiesta. `locale` (*cadena*): si es nacional, regional o local. `locale_name` (*cadena*): la zona a la que afecta la festividad. `description` (*cadena*): descripción de la festividad. `transferred` (*booleano*): si la festividad se ha pasado a otra fecha.

– *Items:* Todos los productos que se venden en la cadena de supermercados. De aquí filtraremos los productos que sean perecederos.

Consta de 4.100 filas.

Campos: `item_nbr` (*entero*): el identificador del producto. `family` (*cadena)*: el tipo de producto. `class` (*entero*): la clase a la que pertenece. `perishable` (*booleano*): si se trata o no de un producto perecedero.

– *Oil:* El precio del barril de petróleo en dólares en cada fecha. No todas las fechas tienen precio.

Consta de 1.218 filas.

Campos: `date` (*fecha*): la fecha del precio. `dcoilwtico` (*real*): el precio en dólares del barril en esa fecha.

– *Stores:* Descripción de las tiendas de la cadena.

Consta de 54 filas.

Campos: `store_nbr` (*entero*): identificador de la tienda. `city` (*cadena*): ciudad en la que está la tienda. `state` (*cadena*): estado federal en el que se encuentra la tienda. `type` (*carácter*): tipo de tienda, de la *A* a la *E*. `cluster` (*entero*): clúster de la tienda (véase una agrupación de tiendas similares).

– *Transactions:* Número de ventas totales por día que se ha producido en cada tienda.

Consta de 83.488 filas.

Campos: `date` (*fecha*): fecha de venta. `store_nbr` (*entero*): identificador de la tienda. `transactions` (*entero*): número de ventas.

– *Train:* Número de artículos vendidos por fecha, tienda y producto.

Consta de 125.497.041 filas.

Campos: `id` (*entero*): identificador de la fila. `date` (*fecha*): fecha de la transacción. `store_nbr` (*entero*): identificador de la tienda. `item_nbr` (*entero*): identificador del producto. `unit_sales` (*real*): número de productos vendidos; los números negativos indican devoluciones. `onpromotion` (*booleano*): si el artículo está en promoción.

### 3.2.3. Exploración

Para realizar la exploración del conjunto de datos y tener una mejor idea del conjunto con el que estamos trabajando, vamos a utilizar la IDE RStudio. A través de un script de R, se han procesado los datos y se han generado las gráficas que se detallan a continuación.

Empezamos con los valores booleanos que disponemos.

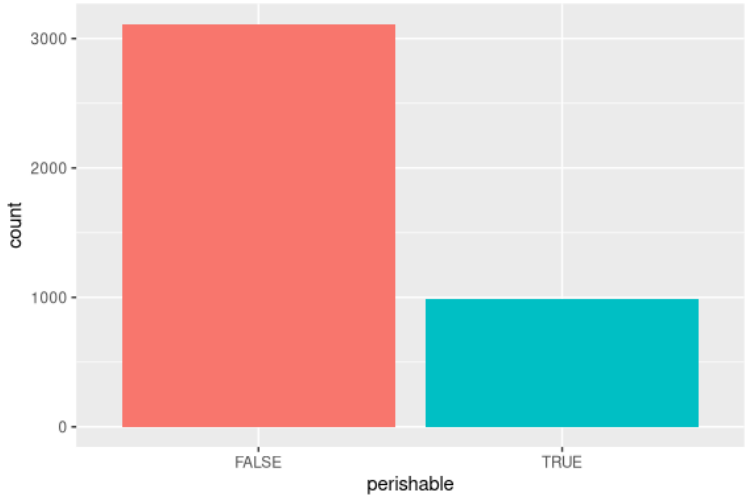


Figura 6: Gráfica con el total de elementos perecederos y no perecederos.

En el caso de los alimentos perecederos, en la figura 6 vemos como la gran mayoría de los alimentos del conjunto de datos son alimentos no perecederos. En nuestro caso, nos quedaremos exclusivamente con los alimentos perecederos, que son los que necesitamos para el modelo de negocio.

También nos encontramos con que la mayoría de productos no están en promoción, como se ve en la figura 7.

En la parte de la izquierda podemos ver una inconsistencia al haber productos sobre los que no se dice si están en promoción. Lo lógico en estos casos es asumir que no están en promoción, y tras procesarlo de esa manera nos queda la gráfica de la derecha.

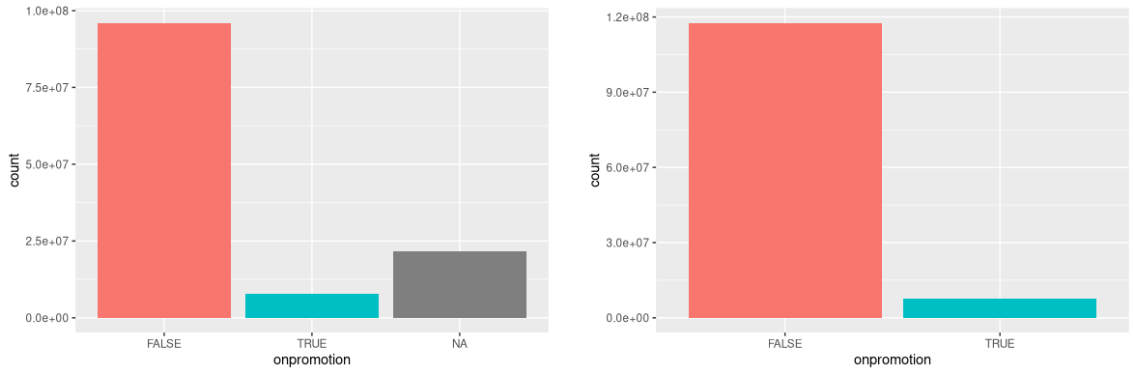


Figura 7: Gráficas con el total de elementos en promoción y sin promoción. A la izquierda con los valores sin procesar y a la derecha procesados.

A partir de aquí nos quedamos exclusivamente con los productos perecederos, y los datos y las gráficas que vamos a ver no tienen en cuenta los productos no perecederos.

Una vez hecha esta selección, vamos a empezar en la figura 8 con los tipos de alimentos que hay en el conjunto de datos y su distribución según las ventas.

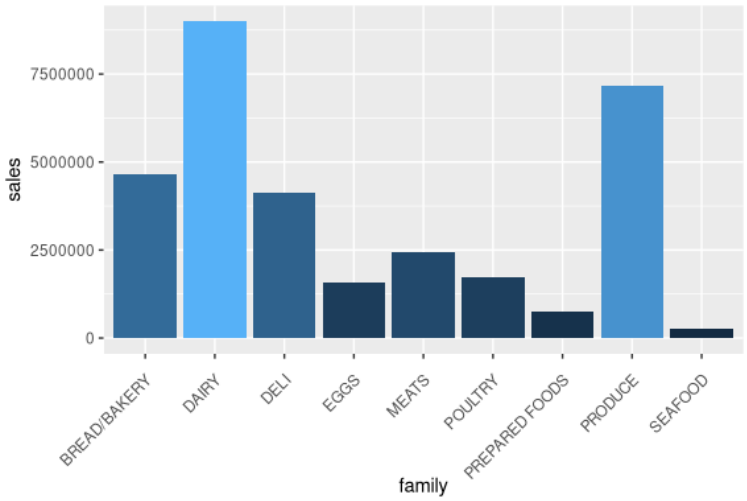


Figura 8: Gráfica con el total de ventas por tipo de alimento.

Aquí vemos que los productos perecederos se distribuyen en tan solo nueve familias de alimentos, siendo las dos más vendidas la de lácteos y la de frutas y verduras.

Pasamos ahora al total de ventas por unidades temporales.

Se aprecia en la figura 9 que la distribución de ventas es bastante homogénea salvo en la distribución por años, donde las ventas han ido incrementando año tras año (el último año, 2017, no lo contabilizamos en esa subida al no tener los datos completos, sino solamente hasta el 15 de agosto).

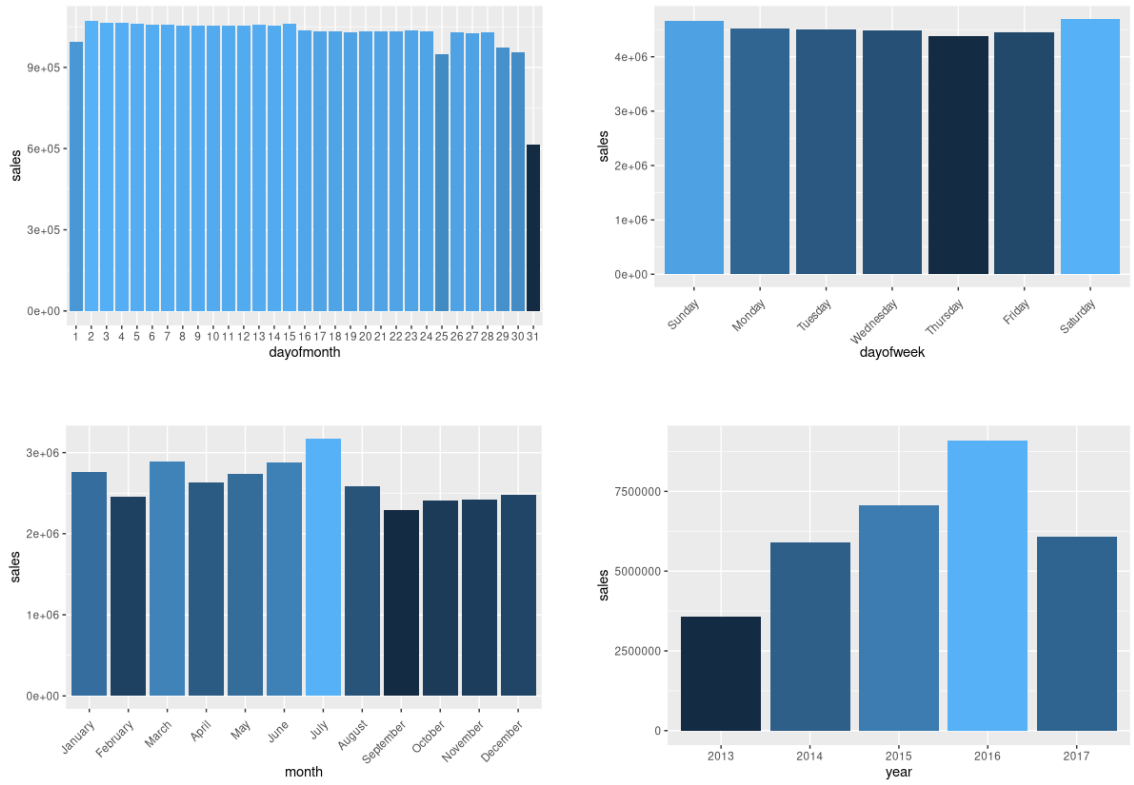


Figura 9: Gráficas con el total de ventas por unidad cronológica. De izquierda a derecha y de arriba a abajo: Ventas por días del mes, por días de la semana, por mes y por año.

Esto nos da una indicación de que los valores de día de la semana, día del mes y mes del año probablemente no tengan mucha influencia a la hora de predecir el comportamiento de las ventas en esta cadena, si bien habría que hacer la comprobación producto a producto.

Por último, en la figura 10 vamos a ver cómo influye la situación territorial de las tiendas en la distribución de ventas. Para ello, hemos comparado el número de tiendas en cada ciudad y estado con el número de ventas en esos territorios.

Lo primero que salta a la vista comparando las gráficas de la izquierda con la derecha es la alta similitud que tienen. Hay una correlación muy alta entre el número de tiendas en una zona territorial y el número de ventas en esa zona.

Lo segundo, es que las gráficas de ciudad y estado son muy similares. No hay grandes variaciones entre una y otra, y es posible que la columna de estado solamente nos dé información redundante ya contenida en la de ciudad.

### 3.2.4. Calidad

Vamos a explorar algunos de los problemas del conjunto de datos que nos hemos encontrado durante la fase de comprensión:

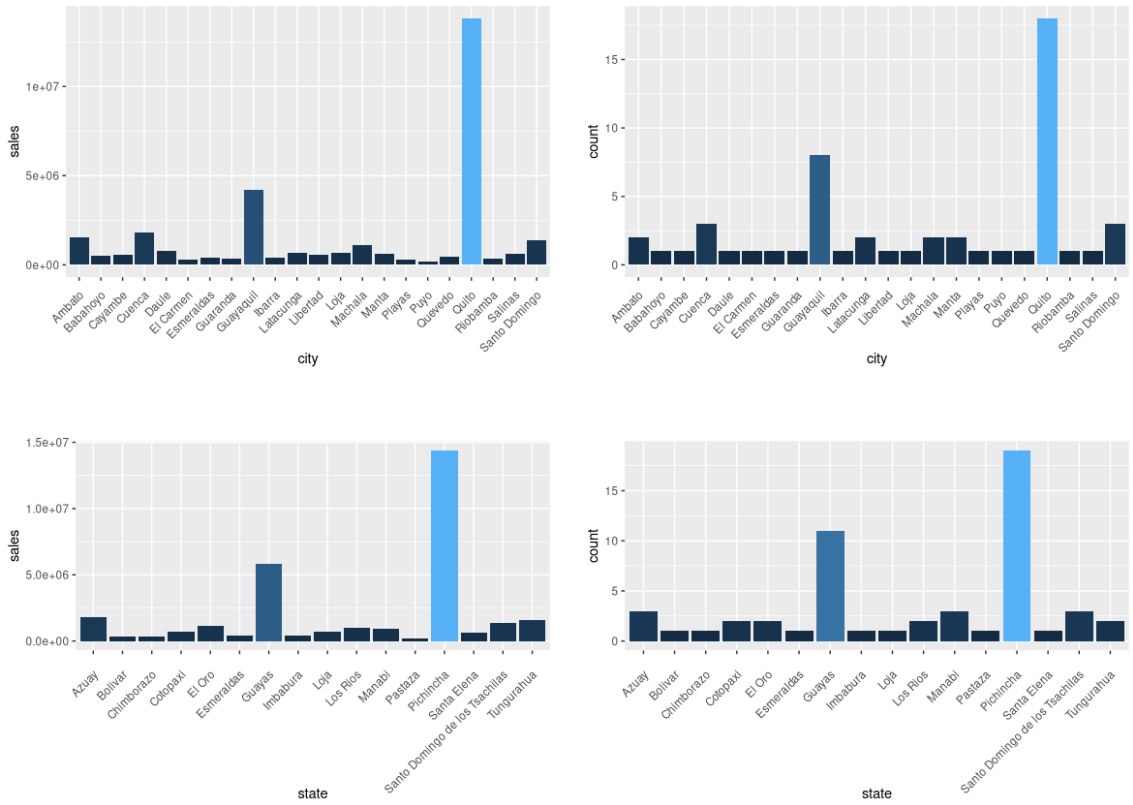


Figura 10: Gráficas que describen la distribución territorial de las ventas y las tiendas. De arriba abajo y de izquierda a derecha: ventas por ciudad y por estado, y número de tiendas por ciudad y por estado.

– *Datos incompletos:* Nos hemos encontrado con que no tenemos todos los precios de barril de petróleo para las fechas de la base de datos. También hemos visto que el campo de los artículos en promoción está vacío en muchas filas.

– *Inconsistencias:* Hemos detectado que en el conjunto del número de artículos vendidos, se han dado en contadas ocasiones número reales, en lugar de enteros, como número de unidades vendidas. Es decir, se han mezclado unidades vendidas con kilos o litros vendidos.

## 3.3. Preparación de los datos

La fase de preparación de datos cubre todas las actividades con las que se construye el conjunto final de datos a partir de los datos iniciales de los que disponemos.

### 3.3.1. Selección

En cuanto a las filas que vamos a seleccionar, en primer lugar, solo nos vamos a quedar con aquellos productos que sean perecederos, y dentro de estos, los que las unidades vendidas se midan con números enteros mayores que cero (por simplicidad vamos a obviar las devoluciones y las mediciones en kilos o litros). Además, para utilizar un conjunto de datos lo más parecido posible al de *Calibrated Model-Based Deep Reinforcement Learning*, vamos a seleccionar todas las fechas que vayan desde 2014-01-01 a 2016-08-31 y los 100 artículos perecederos más vendidos.

En lo que respecta a las columnas, vamos a omitir todas aquellas que sean descripciones, así como la del precio del barril de petróleo, pues es un dato que no conocemos a futuro, y el identificador de transacción, que no nos da información útil.

### 3.3.2. Limpieza

En esta tarea vamos a tratar de hacer frente a los errores que nos encontramos en la tarea de *Calidad*:

- *Datos incompletos:* Para los precios del barril de petróleo, cogeremos el precio del día anterior en los que falten. Y en el caso de los artículos en promoción, tomaremos como *falso* aquellos que aparezcan vacíos.

- *Inconsistencias:* Los productos que tengan un número con decimales de unidades vendidas serán desechados para evitar inconsistencias.

### 3.3.3. Construcción de nuevos datos

Añadiremos los siguientes campos con información derivada de los datos disponibles:

- Media de ventas históricas de los últimos 4, 7, 14 y 28 días.

- Indicador binario sobre el día del mes y la semana del año.

- Funciones cíclicas del seno y coseno del número de días del año transcurrido.

  Es importante codificar los valores cíclicos, como pueden ser los días del año, para hacer saber a nuestros algoritmos de aprendizaje automático que esos valores ocurren en ciclos (Van Wyk, 2022). Una forma común de codificarlos es usando el seno y el coseno de la siguiente manera:

$$x_{sin} = sin(\frac{2\pi x}{max(x)})$$

$$x_{cos} = cos(\frac{2\pi x}{max(x)})$$

Al utilizar esos dos valores, nos queda algo parecido a lo que se muestra en la figura 11, y por lo tanto, se encapsula correctamente la información de que es un valor cíclico.

– Indicador binario sobre si se trata de un día festivo.

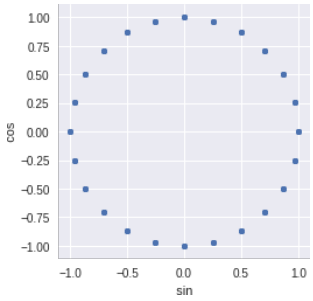


Figura 11: Gráfica con los valores de $x_{sin}$ y $x_{cos}$ en cada eje (Van Wyk, 2022)

### 3.3.4. Integración

Partiremos del conjunto *Train*, con el número de artículos vendidos por fecha, tienda y producto, y le uniremos el resto de datos de los demás conjuntos utilizando los identificadores disponibles.

Con el identificador de tienda, añadiremos los campos relevantes de *Stores* como son la ciudad, el estado, el tipo y el clúster.

A través de la fecha, añadiremos si se trata de un festivo para la tienda.

Y del conjunto *Items* añadiremos la familia y la clase del producto.

### 3.3.5. Formato

En principio, para tener un formato de datos coherente, pasaremos a sustituir todos los valores booleanos por *0* y *1* respectivamente, en lugar de los valores de cadena *True*, *False* o vacío que se utilizan hasta el momento.

Las columnas con los datos correspondientes al tipo de tienda y a la familia de cada producto se procesarán utilizando la codificación *one-hot* donde las distintas cadenas de cada columna se usarán como nuevas columnas con valores binarios. Para evitar el problema de la trampa de la variable binaria (Liaquat, 2020), se va a usar una columna menos del número de elementos distintos de cada columna.

El resto de datos mantendrán el formato original.

## 3.4. Modelado

En esta fase es donde se seleccionan las técnicas de modelado, y se aplican y ajustan sus parámetros para obtener los valores óptimos.

### 3.4.1. Técnica de modelado

Para la técnica de modelado se va a utilizar un sistema basado en agentes que aplique técnicas de aprendizaje automático, tal y como se expone en *Synergistic Integration Between Machine Learning and Agent-Based Modeling: A Multidisciplinary Review* (Zhang y cols., 2021).

El AA se utilizará a nivel de micro-agente; en concreto, se utilizará para predecir el número de ventas de cada artículo y en cada tienda para el día siguiente.

En primer lugar, vamos a desarrollar un sistema basado en agentes que coja los valores de entrada de la base de datos *Corporación Favorita* y haga predicciones aleatorias. Esto nos permitirá ver posibles problemas y mejoras que puedan surgir al realizar el modelo.

Posteriormente se va a utilizar la técnica de aprendizaje supervisado *Random Forest* (Ho, 1995). De esta forma, permitiremos que nuestro sistema admita algoritmos de aprendizaje supervisado.

Luego, se implementará una red neuronal recurrente (RNR), como la utilizada para el mismo problema en *Accurate Uncertainties for Deep Learning Using Calibrated Regression* (Kuleshov, Fenner, y Ermon, 2018). Y sobre esta red neuronal se empleará una técnica de aprendizaje por refuerzo profundo basado en modelos calibrados similar a la descrita en *Calibrated Model-Based Deep Reinforcement Learning*; para ello utilizaremos la técnica de control: control predictivo por modelo (CPM), además del algoritmo de calibración para problemas de regresión de *Accurate Uncertainties for Deep Learning Using Calibrated Regression* (Kuleshov, Fenner, y Ermon, 2018). Con esto, permitimos que el sistema también acepte distintos algoritmos de ARP.

En último lugar, se va a trabajar con una red neuronal prealimentada (RNP) que sea lo suficientemente sencilla y no necesite calibrarse, para demostrar que un sistema que no sea preciso, aunque esté calibrado, no va a dar buenos resultados.

### 3.4.2. Test de diseño

Se hará una separación del conjunto de datos en un conjunto de entrenamiento y otro de prueba en base a un rango de fechas. Específicamente, el subconjunto de datos que va desde 2014-01-01 a 2016-05-31 se usará como conjunto de entrenamiento, y el que va desde 2016-06-01 a 2016-08-31 como conjunto de prueba.

Al tratarse fundamentalmente de un problema de regresión, utilizaremos el porcentaje de error en la estimación como medida para valorar la eficacia del modelo. Se estimarán como válidos los porcentajes de error presentados en el apartado de *Comprensión de negocio*.

### 3.4.3. Construcción del modelo

Vamos a pasar a describir el **modelo basado en agentes** de nuestro trabajo, que se puede ver en la figura 12.

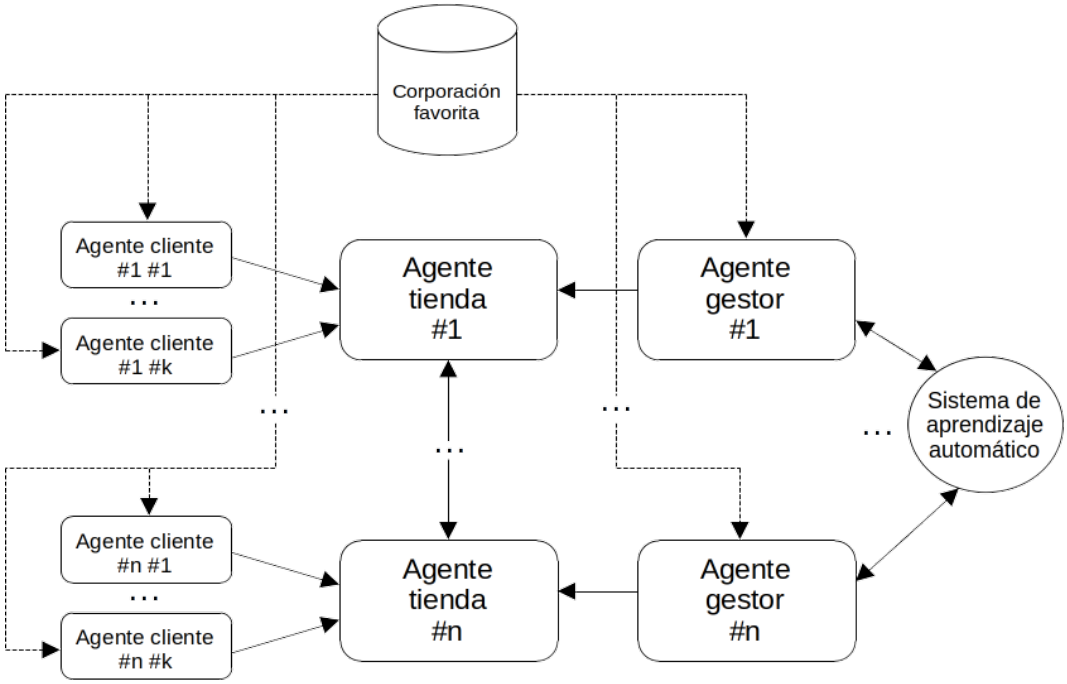


Figura 12: Modelo basado en agentes de nuestro gestor de inventario.

Nuestro sistema cuenta con tres tipos diferentes de agentes:

– *Clientes:* Se trata de agentes proactivos. Simulan el comportamiento de los clientes basándose por completo en el conjunto de datos.

  El número de clientes por tienda viene determinado por el usuario. Cada cliente hará una compra con un número de unidades aleatorias por cada producto, de cero al número de unidades que reste de ese producto. El último cliente hará una compra de todo lo restante.

  Al no tener datos de compras específicas por horas, con una ejecución aleatoria y un número de clientes alto, podemos hacer una simulación lo más parecida posible al mundo real.

– *Gestores:* También se trata de agentes proactivos. Se encargan de pedir las unidades de cada producto para el día siguiente.

  Utilizan un sistema de aprendizaje automático para predecir la demanda en el día siguiente y la cantidad de productos a pedir según el stock disponible. Con esa información, hacen las peticiones de inventario necesarias. Básicamente, su comportamiento viene controlado por el sistema de aprendizaje automático.

– *Tiendas:* En este caso son agentes fundamentalmente reactivos. Son las encargadas de llevar el inventario y hacer las anotaciones de las compras y ventas de los clientes.

  Entran en funcionamiento principalmente cuando cualquiera de los otros dos agentes interactúa con ellos:

  Cada vez que un agente gestor hace una predicción, el agente tienda lo anota para obtener el inventario del día siguiente al acabar la jornada.

  Cuando un agente cliente hace una compra, el agente tienda lo anota y hace la comprobación del stock. En caso de que se supere un umbral de stock vendido marcado por el usuario, el agente intentará transportar stock disponible de otras tiendas de la misma ciudad.

De esta forma, la micro-estrategia (pedir productos para el día siguiente) vendrá marcada fundamentalmente por el sistema de aprendizaje automático, mientras que la macro-estrategia (la gestión de la tienda) la llevará a cabo el sistema basado en agentes.

Tanto el transporte del inventario para el día siguiente como el que se hace entre las tiendas por falta de stock se modelará de forma muy simplificada, no teniendo en cuenta ningún tipo de problema, retraso, coste o penalización.

En la figura 13 se puede observar una simulación en la plataforma Mesa de nuestro modelo, utilizando datos aleatorios en lugar de un sistema de aprendizaje automático. En la ejecución, se va simulando cada día del periodo de prueba.

Todos los agentes del sistema se han implementado como una subclase de la clase `Agent` de Mesa.

El primer día de la simulación, como el stock lo suponemos vacío, solamente se piden los productos para el día siguiente. De ahí que en todas las gráficas que aparecen en este trabajo, el primer día siempre aparezcan a cero los desperdicios y las roturas de stock.

Para cada uno de los siguientes días, primero entran en acción los agentes clientes, que van haciendo pedidos de forma aleatoria hasta llegar a la cantidad marcada por el conjunto de datos. En caso de que se permita el transporte entre tiendas, cuando el stock de un producto esté por debajo de un mínimo, el agente de la tienda correspondiente se

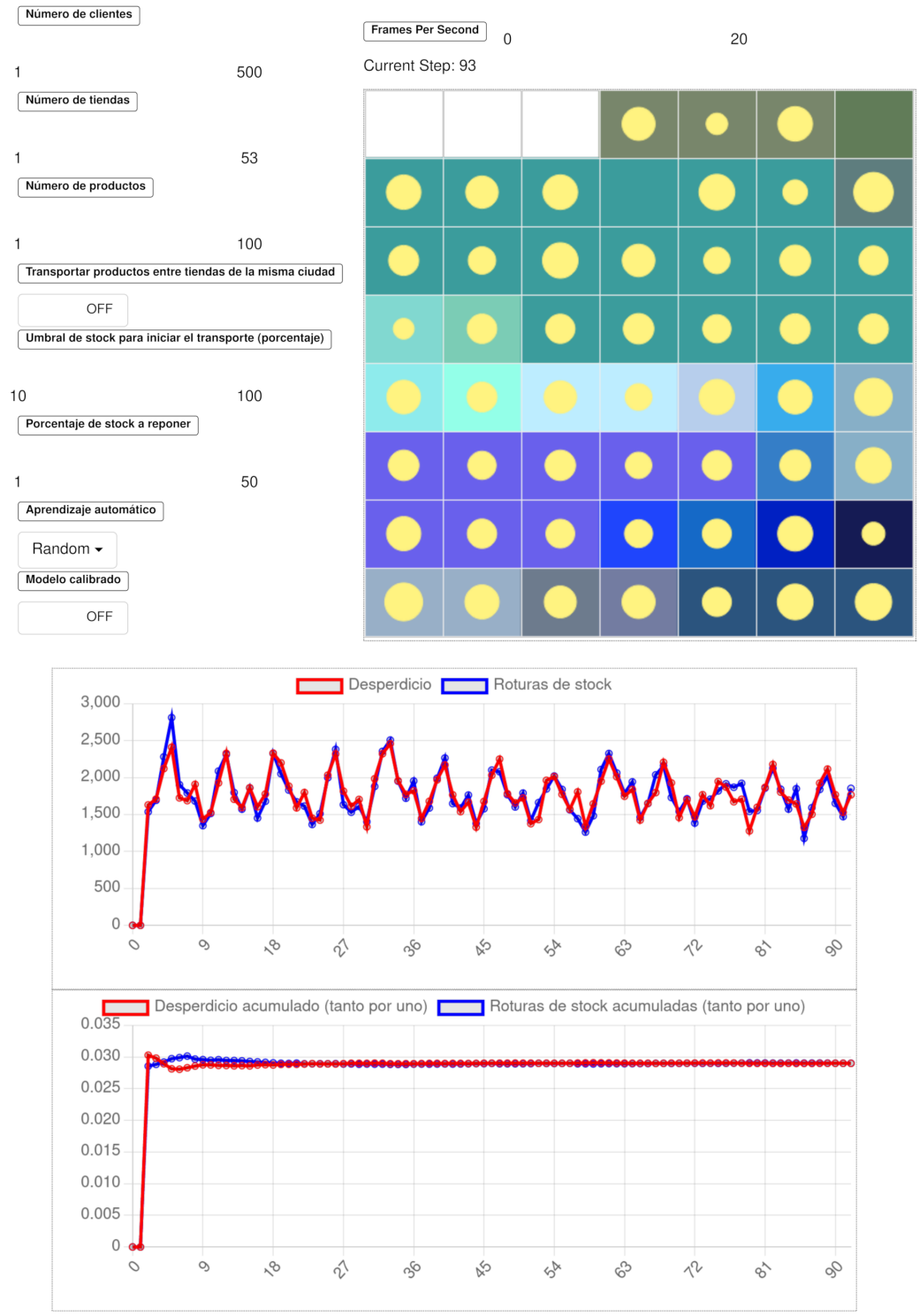


Figura 13: Simulación basada en agentes de nuestro gestor de inventario, sin utilizar AA, con 1 día de caducidad y el máximo número de tiendas y productos.

encargará de transportar productos desde otras tiendas con stock que estén en su misma ciudad.

Cuando todos los agentes clientes han terminado, llega el turno de los agentes gestores,

que hacen las predicciones para el día siguiente. Para la simulación de la figura 13, se multiplica la demanda real del día siguiente por una muestra positiva de una distribución normal $\mathcal{N}(1, \sigma^2)$, siendo $\sigma$ el ruido que queremos aplicar a la demanda real. Cabe decir que los algoritmos de AA en ningún momento hacen uso de las demandas reales a la hora de hacer sus predicciones.

En último lugar, se cierra el día haciendo recuento de los desperdicios de comida y las roturas de stock que ha habido en cada tienda, de cara a mostrar las gráficas correspondientes. Los artículos que se habían pedido para el día siguiente se dan por entregados y se calcula el nuevo stock para continuar con el siguiente paso.

En la cuadrícula que se ve, las tiendas van ordenadas por ciudad y estado, y las que están situadas en la misma ciudad tienen el mismo color; los círculos amarillos denotan la suma de desperdicios y roturas de stock de cada tienda en la fecha de la ejecución. La simulación también incluye las gráficas de desperdicio y roturas de stock, tanto por día como el porcentaje acumulado. Además, se pueden controlar los distintos parámetros de la simulación, como el número de clientes, de tiendas, productos, caducidad de los productos, el sistema de aprendizaje automático empleado, si queremos que los agentes tienda puedan transportar productos entre otras tiendas de la misma ciudad, etc.

Para más información sobre el uso del simulador, se puede consultar la *Guía de usuario*.

Una vez que tenemos claro el modelo basado en agentes, viene la hora de integrarlo con un sistema de AA.

Como expusimos anteriormente, comenzaremos integrándolo con el método ***Random Forest***.

Durante la implementación e integración, nos damos cuenta de que el módulo encargado de enlazar con el sistema de aprendizaje automático en el agente gestor podía implementarse de forma más eficiente para que la simulación fuera más fluida; cosa que hicimos.

En cuanto a los parámetros de *Random Forest*, nos fijaremos en los dos parámetros más importantes: número de elementos (árboles) y profundidad máxima. Nos hemos centrado solo en estos dos parámetros porque tanto el tiempo de ejecución como la cantidad de recursos necesarios para obtener los valores más eficientes para nuestro conjunto de datos han sido bastante elevados.

Comenzamos utilizando los parámetros estándares (100 árboles y sin profundidad máxima), y nos encontramos con que el entrenamiento y la predicción son muy lentos, y con que el fichero en el que se guarda el entrenamiento es bastante grande (supera los 3GB), por lo que su carga a memoria también es muy lenta, haciendo en definitiva que el sistema sea prácticamente inviable. De esta manera nos planteamos optimizar los dos

valores anteriores. Para ello, vamos a seguir las recomendaciones de *In Depth: Parameter tuning for Random Forest* (Ben Fraj, 2017).

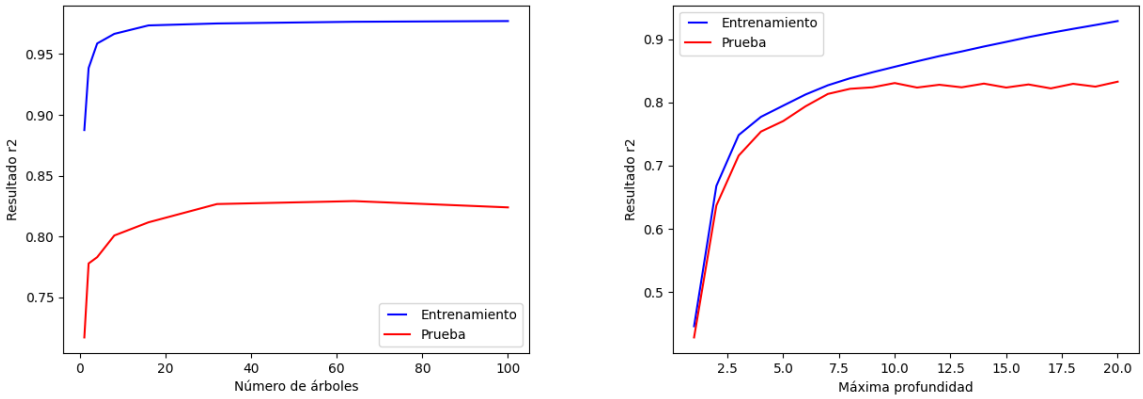


Figura 14: Coeficiente de determinación del conjunto de entrenamiento y de prueba para distintos valores de número de árboles (izquierda) y profundidad máxima (derecha).

Se ha utilizado el coeficiente de determinación ($R^2$) para calibrar cómo de eficaz es el algoritmo con el cambio de parámetros. Se han usado distintos valores de números de árboles (1, 2, 4, 8, 16, 32, 64 y 100) y de profundidades máximas (de 1 a 32), obteniendo los resultados de la figura 14.

Se puede observar que para el número de árboles el valor adecuado está en torno a los 32, dado que un mayor número no incrementa la eficacia del sistema. En cuanto a la profundidad máxima, el valor ideal se encuentra en los 10, ya que a partir de ese número se empieza a ver un sobreajuste donde decae la eficacia del sistema.

Con los valores que hemos obtenido, el entrenamiento del sistema de AA se puede guardar en un fichero de 1.5MB (más de 2000 veces más pequeño que con los parámetros por defecto), y su ejecución es fluida.

Ahora vamos a continuar con el siguiente paso de modelado.

Se va a utilizar una **red neuronal recurrente** para estimar la demanda diaria de productos. Las dos RNR que hemos considerado han sido GRU, que es la utilizada en *Accurate Uncertainties for Deep Learning Using Calibrated Regression*, y LSTM. Tras probar ambas con distintos parámetros se puede ver en la figura 15 que ambos obtienen un coeficiente de determinación muy similar en los parámetros óptimos, por lo que decidimos seguir adelante con ambos modelos. Se ha elegido para ambos como parámetro de tamaño de lotes 100, que es el que obtiene un mayor coeficiente $R^2$.

Para hallar un modelo probabilístico de estas RNR se ha utilizado la técnica *MC dropout*. Se han usado 50 muestras por cada predicción para obtener la distribución de

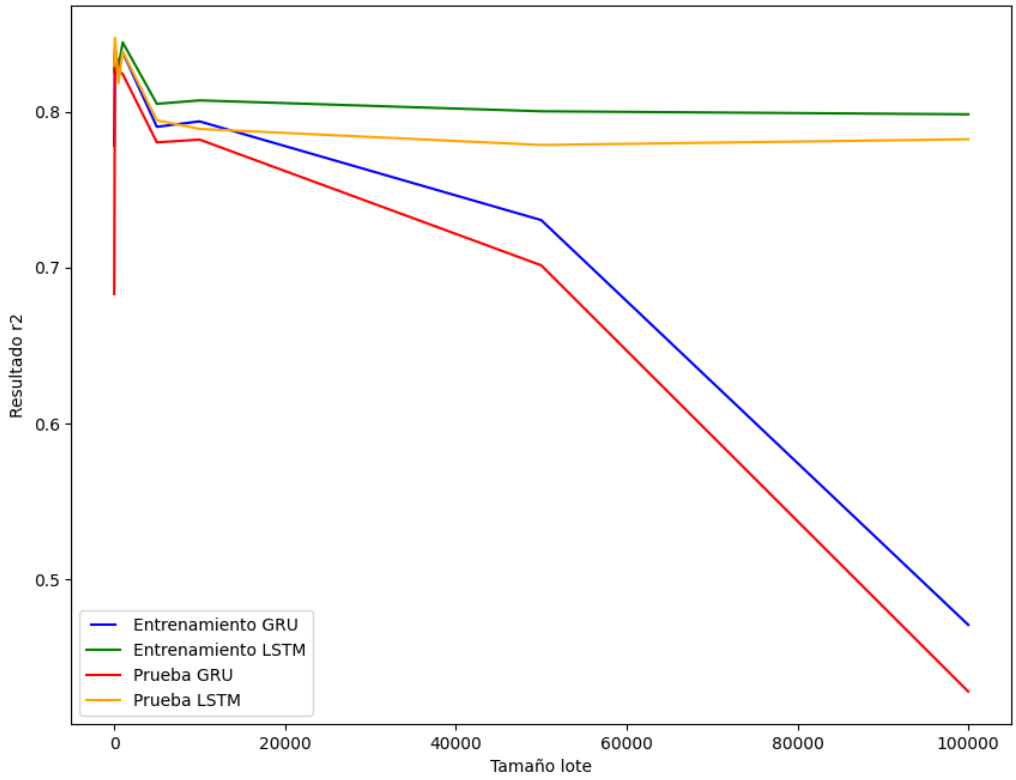


Figura 15: Coeficiente de determinación del conjunto de entrenamiento y de prueba para distintos valores de tamaños de lotes para LSTM y GRU.

probabilidad de la salida. La calibración de la predicción la realizamos con el algoritmo descrito para regresión en *Accurate Uncertainties for Deep Learning Using Calibrated Regression* (Kuleshov, Fenner, y Ermon, 2018).

Este modelo probabilístico sirve de base para la resolución del problema a través de aprendizaje por refuerzo profundo, donde definimos el siguiente modelo:

Formalizamos el problema de la gestión de inventario para un alimento como un PDM $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$; los estados $s \in \mathcal{S}$ son tuplas $(d_s, q_s)$, donde $d_s$ es el día del calendario y $q_s \in \mathbb{Z}^L$ es un vector de longitud $L$ (caducidad máxima) en el que el elemento i-ésimo representa el número de alimentos en stock que caducan en i días (Malik y cols., 2019). La función de transición $\mathcal{P}$ se define de la siguiente manera: cada día se produce una demanda de $n$ unidades; al final del estado de transición, se eliminan los $n$ productos demandados del stock, se desechan los que caducaran al día siguiente, y se añaden los nuevos productos pedidos al inventario. Las acciones $a \in \mathcal{A} \subset \mathbb{Z}$ corresponden a la cantidad de unidades pedidas para el día siguiente. Y las recompensas $r \in \mathcal{R}$ se calculan como la suma de los productos desechados y las roturas de stock.

Para el aprendizaje por refuerzo profundo utilizamos la técnica de control CPM, donde usamos 5.000 trayectorias aleatorias sobre un horizonte de $L$-pasos, siendo $L \in \mathbb{Z}$ la caducidad máxima en días de los productos, de donde escogemos el primer paso de la trayectoria con mayor recompensa estimada.

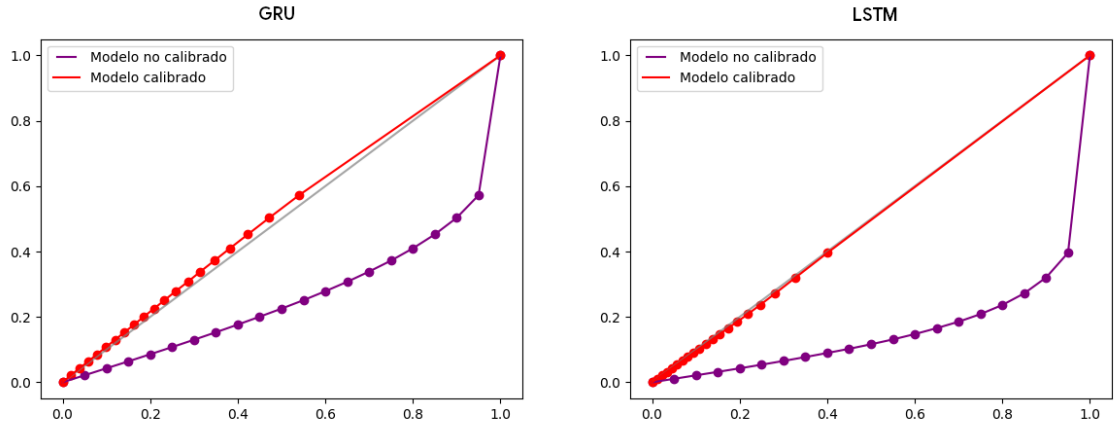


Figura 16: Curvas de calibración para GRU (izquierda) y LSTM (derecha).

Además, utilizamos la técnica de calibración mencionada en la *técnica de modelado*: Con un conjunto de calibración no entrenado, obtenemos los niveles de confianza estimados para distintos cuantiles $k$, con $k \in [0, 1]$ tomando valores como 0, 0,1, 0,2, ..., 1. Para cada cuantil calculamos también el nivel de confianza observado, sumando el número de elementos cuyo valor etiquetado está dentro del nivel de confianza estimado. Finalmente, usamos regresión isotónica para transformar los niveles de confianza estimados en niveles de confianza observados.

En la figura 16 podemos ver las curvas de calibración de las dos RNR.

Para obtener predicciones en los modelos calibrados, se han utilizado muestras aleatorias dentro de un intervalo de confianza calibrado del 85 %.

En último lugar, utilizaremos una **red neuronal prealimentada** que no necesite calibración. La figura 17 deja ver que el modelo está perfectamente calibrado de inicio. Este modelo, sin embargo, no tiene un buen coeficiente de determinación comparado con las RNR anteriormente detalladas, de forma que las predicciones que hace no son nada precisas.

### 3.4.4. Valoración

En este apartado expondremos una valoración simplificada de los distintos modelos. Una valoración más exhaustiva se puede encontrar en el apartado de *Estudio de resultados*.

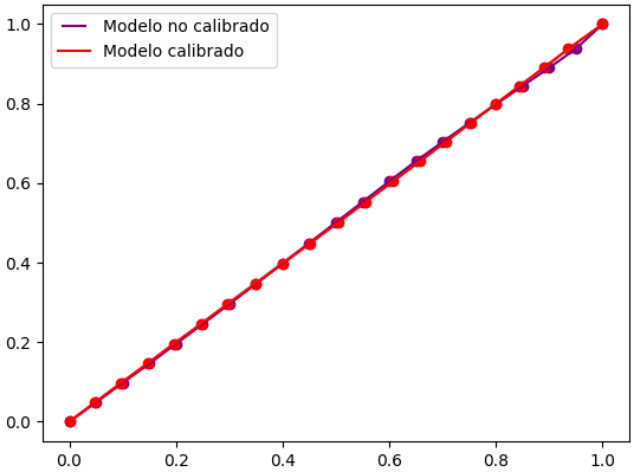


Figura 17: Curva de calibración de nuestra RNP.

Tras valorar los dos primeros modelos (*Random Forest* y RNR), nos encontramos con que ambos cumplen con los tests establecidos en la fase de modelado para el parámetro de días de caducidad suficientemente alto. Cuanto más sean los días de caducidad, mejor es el comportamiento del sistema, puesto que puede compensar las sobreestimaciones con las infraestimaciones y reducir los desperdicios y las roturas de stock. Por lo tanto, ambos modelos se consideran aceptables para pasarlos a la siguiente fase.

Con el número de ciudades y de productos al máximo, la simulación utilizando tanto el método CPM como las RNB es bastante lenta, y la combinación de ambas técnicas hace que la misma no sea viable. Para evaluar el sistema en estas condiciones se ha reducido tanto el número de ciudades como el de productos, para poder obtener las métricas correspondientes.

El parámetro de reentrenamiento produce consistentemente significativamente peores resultados para todos los algoritmos probados y con distintos lotes de fechas de reentrenamiento. El parámetro se mantendrá en esta versión con fines de documentación, y se propone suprimirlo en posteriores versiones al no cumplir con los objetivos de la ciencia de datos.

En las pruebas realizadas la calibración mejora levemente los resultados obtenidos. La calibración por sí sola claramente no es suficiente: los resultados obtenidos con el último modelo (RNP) no cumplen los objetivos de negocio.

## 3.5. Evaluación

Al llegar a esta etapa, ya se ha determinado que los modelos planteados en la etapa anterior cumplen los objetivos de la ciencia de datos. Ahora corresponde hacer una evaluación del sistema teniendo en cuenta los objetivos de negocio.

### 3.5.1. Evaluación de resultados

En nuestro caso, los objetivos de negocio son bastante claros y concisos: se trata de reducir el desperdicio de comida y las roturas de stock en una cadena JIT de alimentación.

Tras evaluar distintas simulaciones del sistema, podemos llegar a la conclusión de que se cumplen los objetivos de negocio que propusimos al principio del trabajo. Los modelos consiguen ajustarse a los parámetros marcados, reduciendo considerablemente el desperdicio de comido y roturas de stock.

Tenemos una métrica concisa de suma de desperdicios y roturas de stock que nos permite hacer un ranking objetivo de los modelos implementados, de manera que los que mejor ranking tienen son los que obtienen menor valor en dicha suma.

Con este simulador se da respuesta a la pregunta más importante que una cadena de supermercados puede hacerse: si su gestión de inventarios es eficiente y si le ayudaría que dicha gestión fuera llevada directamente por nuestro sistema de aprendizaje automático.

### 3.5.2. Revisión del proceso

El proceso de revisión nos ha ayudado a comprender los siguientes puntos sobre el desarrollo de un trabajo de ciencia de datos con la metodología CRISP-DM:

- Es importante no perder de vista cuál es la idea de negocio en la parte de modelado. A veces prestamos demasiada atención a los datos y nos olvidamos de que se debe cumplir un objetivo de negocio y de que el modelo debe ir encaminado a satisfacerlo.

  Y una vez terminada la fase de modelado, la idea de negocio va a ser la que va a marcar qué modelo elegir (si alguno) y qué caminos seguir a continuación.

- En la fase de modelado siempre hay que contar con que el ajuste de parámetros va a llevar más tiempo del que pensamos. Cualquier predictor o clasificador cuenta con múltiples parámetros que debemos probar y comparar en un conjunto de datos bastante grande.

- La preparación del conjunto de datos es una tarea que, aunque sencilla, conlleva bastante tiempo. Se suele trabajar con conjuntos de datos bastante grandes para poder

obtener información interesante, y eso hace que se necesiten de muchos recursos de memoria y de tiempo para procesar esos datos.

Es aconsejable empezar con un subconjunto pequeño del conjunto de datos para comprobar que funcionan las manipulaciones que queremos realizar, antes de implementarlas en el conjunto completo.

### 3.5.3. Próximos pasos

Llegados a este punto, consideramos que nuestro sistema es lo suficientemente eficaz y relevante como para pasar a la fase de implantación.

En cualquier caso, también se deja abierto el proceso para volver a la fase de modelado si hiciera falta en caso de que se quieran estudiar nuevos modelos que pudieran cumplir los objetivos de negocio y los de ciencia de datos, y ayuden a mejorar el sistema.

## 3.6. Implantación

En esta fase se utilizará el conocimiento adquirido con el sistema desarrollado para conseguir mejoras en la organización en la que se va a implantar.

### 3.6.1. Plan de implantación

El siguiente trabajo se ha desarrollado sobre un conjunto de datos estadísticos de una cadena de supermercados venezolana.

La primera tarea para una implantación satisfactoria es la de obtener un conjunto de datos real de la cadena de supermercados en la que se vaya a instalar. Este es el primer paso que se dio en el estudio *Towards a Sustainable Food Supply Chain Powered by Artificial Intelligence* (Kuleshov, Seymour, y cols., 2018), donde después de desarrollar el sistema con el mismo conjunto de datos que nuestro sistema, pasaron a implantarlo en una cadena de supermercados estadounidense con más de cien tiendas, obteniendo resultados similares a los de prueba.

También es importante que las personas implicadas dentro de la organización tengan los conocimientos necesarios y estén preparados y concienciados para la implantación. En una entrevista en *PAW Climate 2021* (Work on Climate, 2021), Volodymyr Kuleshov relata cómo su sistema de gestión de inventario fue anulado temporalmente por uno de los gerentes de la cadena cuando el gerente principal estuvo de baja. Por ello, es importante que reciban la información y entrenamiento adecuados:

– *Gerentes:* Los gerentes encargados de inventarios deben ser informados de las recomendaciones del sistema para llevarlas a cabo y que no haya discrepancias entre ellos y el sistema.

– *Personal informático:* El personal informático estará encargado de monitorizar el sistema y comprobar que no haya problemas, y por tanto deben estar informados a nivel técnico de su funcionamiento.

– *Gestores de bases de datos:* Los gestores de bases de datos deben estar informados de la estructura de los datos que requiere el sistema para poder integrar los datos de la cadena relativos al inventario con el sistema.

### 3.6.2. Plan de monitorización y mantenimiento

La tarea más importante de mantenimiento es la de comprobar que el gestor de inventario efectivamente mejora el sistema de la cadena de suministro del supermercado. Para ello, se producirá una monitorización diaria de las decisiones tomadas por el gestor y del impacto que tienen en la cadena.

Se comprobará de forma periódica si los datos sobre los que trabaja el gestor (los datos de entrenamiento) siguen siendo válidos para la predicción actual, y si, en caso de que no lo sea, haría falta entrenar al sistema desde cero con un nuevo conjunto de datos de entrenamiento.

También se hará una comprobación periódica, más espaciada en el tiempo, en la que se compruebe si diferentes modelos a los aquí utilizados mejoran la eficiencia de los modelos actuales, y, en dicho caso, implantar estos nuevos modelos en el sistema.

### 3.6.3. Informe final

Tomaremos el presente trabajo como informe final, donde se evalúa, informa y compara por completo el sistema desarrollado.

### 3.6.4. Revisión del proyecto

En este último punto de la metodología CRISP-DM se formulan las impresiones finales y lecciones aprendidas durante el proceso. Hemos decidido desarrollar este punto en el apartado de *Conclusiones y líneas futuras*.

# 4

# Estudio de resultados

La caducidad de los alimentos es un factor muy importante a la hora de desarrollar un gestor de inventario de alimentos perecederos, como veremos a continuación. Cuanto mayor es la caducidad de los alimentos, mejores resultados proporciona el sistema. Este es un resultado lógico, dado que una mayor caducidad le da también un mayor margen a nuestro gestor para compensar sobreestimaciones con infraestimaciones y obtener un mejor porcentaje acumulado.

En el conjunto de datos con el que hemos trabajado, no había ninguna información acerca de la caducidad de los productos que se vendían, solamente sabíamos si eran perecederos o no. En los tres estudios que han utilizado aprendizaje por refuerzo profundo sobre el mismo conjunto de datos que este sistema, han utilizado una caducidad de 5 días desde que el alimento llega al establecimiento hasta que se desecha. Esos estudios son: *Calibrated Model-Based Deep Reinforcement Learning* (Malik y cols., 2019), *Accurate Uncertainties for Deep Learning Using Calibrated Regression* (Kuleshov, Fenner, y Ermon, 2018) y *Towards a Sustainable Food Supply Chain Powered by Artificial Intelligence* (Kuleshov, Seymour, y cols., 2018). En este trabajo sin embargo se ha decidido que la caducidad de los alimentos fuera uno de los parámetros de la simulación que el usuario puede elegir para poder comprobar el comportamiento en diferentes configuraciones de caducidad.

Dicho esto, vamos a empezar con el estudio de los resultados de los distintos modelos.

Comenzaremos con ***Random Forest***.

La figura 18 nos muestra una simulación completa del sistema usando este algortimo.

Por otro lado, en la figura 19 se observan las gráficas de simulaciones con distintos valores de caducidad. Como ya explicamos en el primer párrafo, a mayor caducidad, mejor

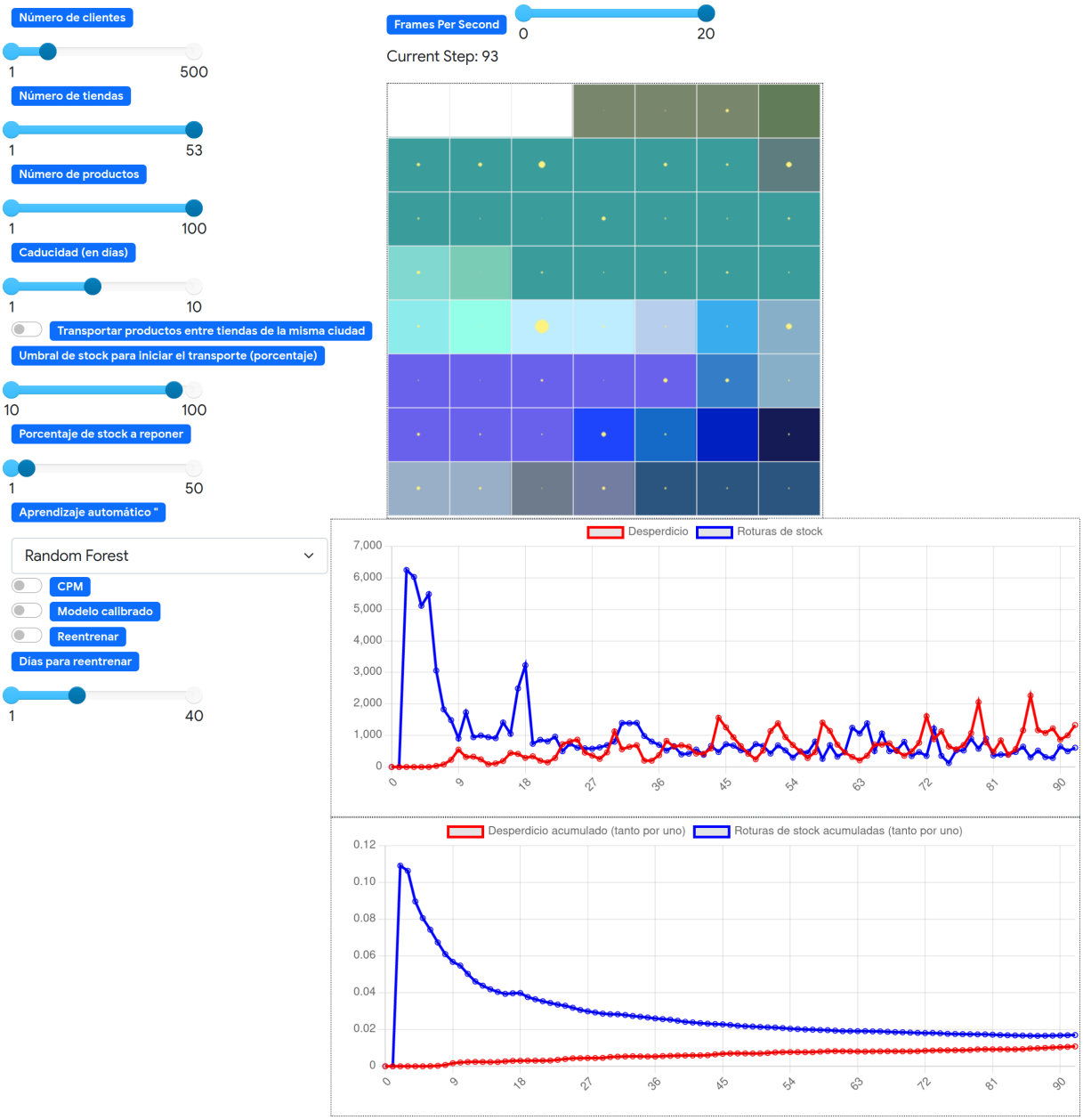


Figura 18: Ejecución del sistema usando el algoritmo *Random Forest* con 5 días de caducidad y el máximo número de tiendas y productos.

rendimiento. Así, para caducidad con valor 1 tenemos que el desperdicio acumulado es del 14,50 % y las roturas de stock del 13,6 %. Si pasamos a caducidad con valor 3, notamos un cambio drástico donde el desperdicio acumulado es del 2,9 % y las roturas de stock del 2,7 %. Conforme vamos aumentando la caducidad, la diferencia se nota menos, y, por ejemplo, con caducidad con valor 10, el desperdicio acumulado es del 1,7 % y las roturas de stock del 0,2 %.

Otro dato interesante que podemos apuntar es que, al aumentar el valor de la caducidad, aumentamos la diferencia entre las roturas de stock y los desperdicios. Como se ve en las gráficas por día, esto se debe a que en los $n$ primeros días (siendo $n$ el valor de la

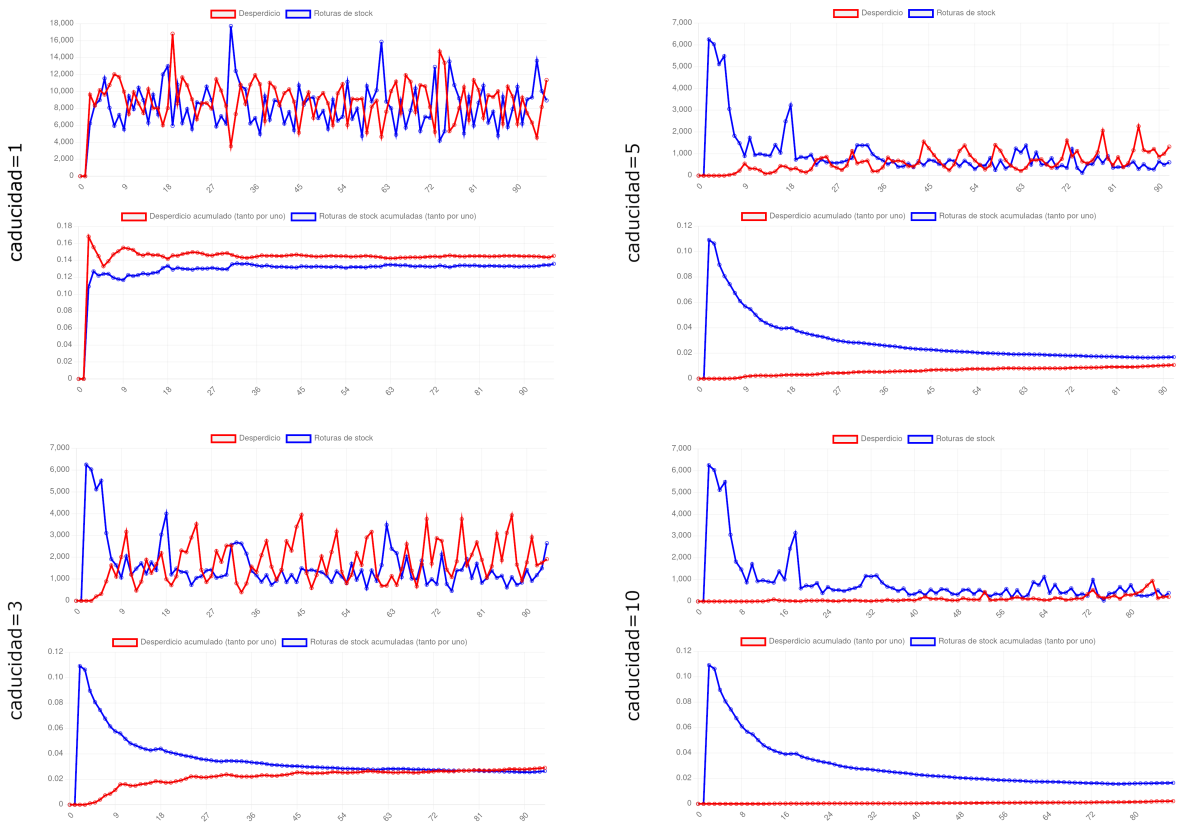


Figura 19: Gráficas de simulación para *Random Forest* con distintos valores de caducidad.

caducidad) no hay elementos que caduquen y sí puede haber roturas de stock.

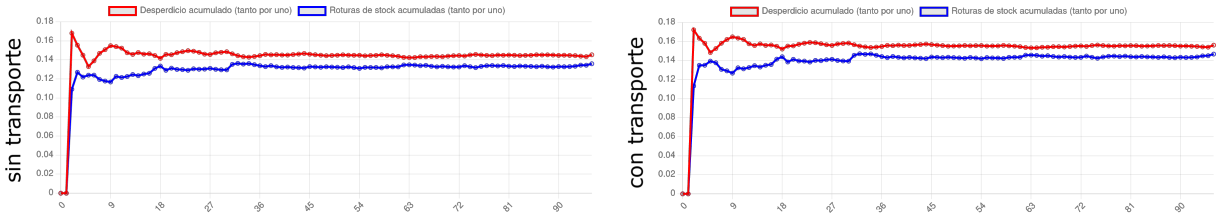


Figura 20: Gráficas de simulación para *Random Forest* con transporte de productos entre tiendas de la misma ciudad (izquierda) y sin transporte (derecha).

En cuanto al transporte entre ciudades, se ha probado con distintos parámetros y en ningún momento mejora los valores obtenidos sin transporte. En la figura 20 se ve un ejemplo en que los valores son ligeramente peores con transporte (15,6 % de desperdicio y 14,7 % de roturas de stock frente a 14,5 % y 13,6 % respectivamente). Hay que tener en cuenta que no se ha modelado ningún tipo de penalización por transportar productos entre tiendas de la misma ciudad, de forma que si se tuviera en cuenta dicha penalización que sí existe en la vida real, los resultados serían aún peores.

Otro dato interesante es que el reentrenamiento, con cualquier valor de días para

reentrenar y cualquier configuración de parámetros, produce sistemáticamente peores resultados.

Pasamos ahora a ver los resultados de las **RNR**.

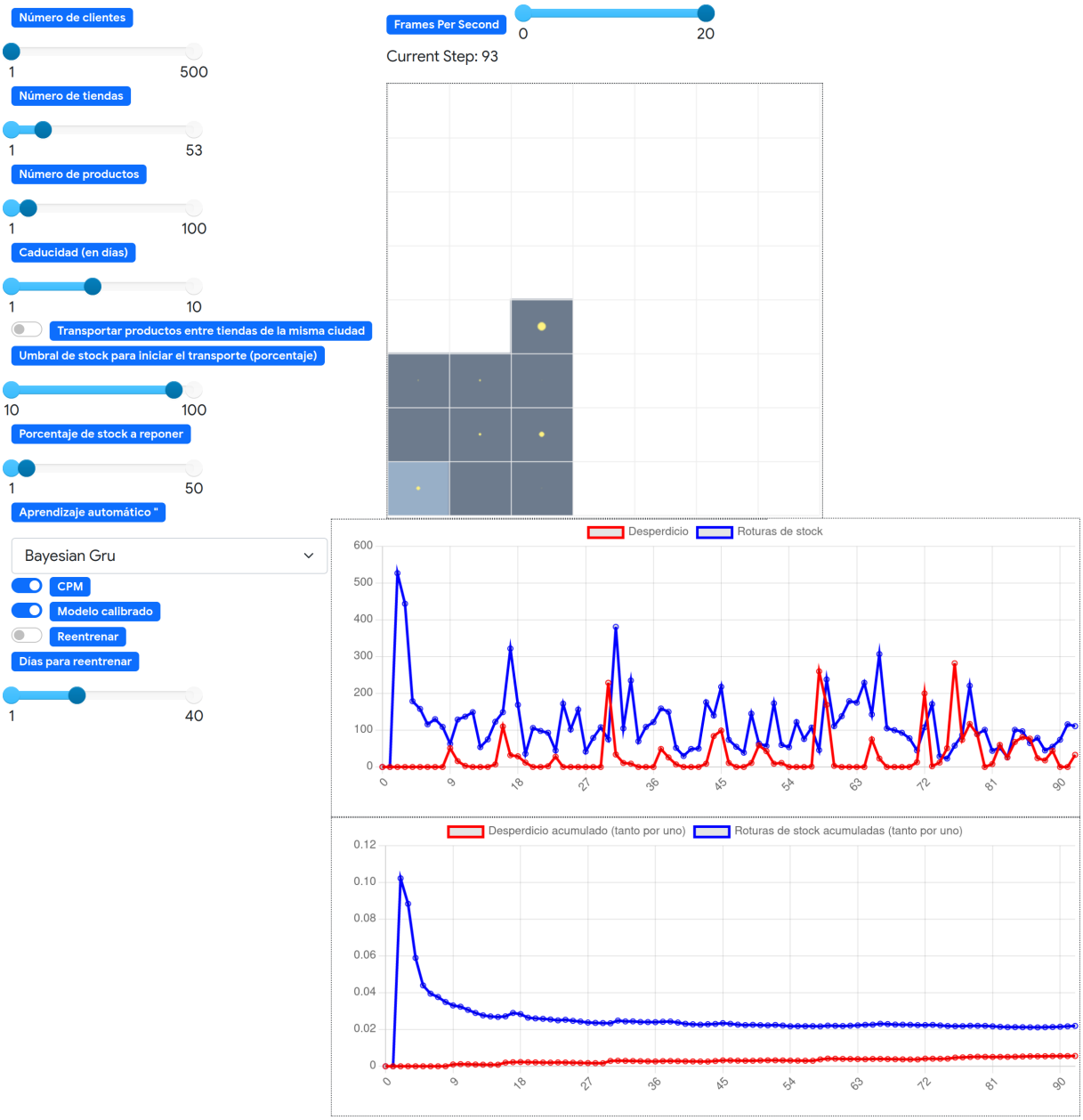


Figura 21: Ejecución del sistema usando una RNR (en particular GRU) con 5 días de caducidad, con CPM y calibración, y utilizando las 10 tiendas con más ventas y los 10 productos más vendidos.

Al igual que hemos hecho con *Random Forest*, mostramos en la figura 21 una simulación completa usando una GRU bayesiana, con CPM y calibración, y con 5 días de caducidad.

Con respecto a la caducidad, tanto en la versión determinista como en la bayesiana nos encontramos con el mismo fenómeno ya explicado que con *Random Forest*: mejoras

muy grandes entre los valores 1 y 2, y conforme aumenta la caducidad, se hacen menores las mejoras.

También hemos observado el mismo comportamiento con el transporte entre ciudades. Los resultados son ligeramente peores siempre que se activa este transporte, independientemente de los demás parámetros elegidos.

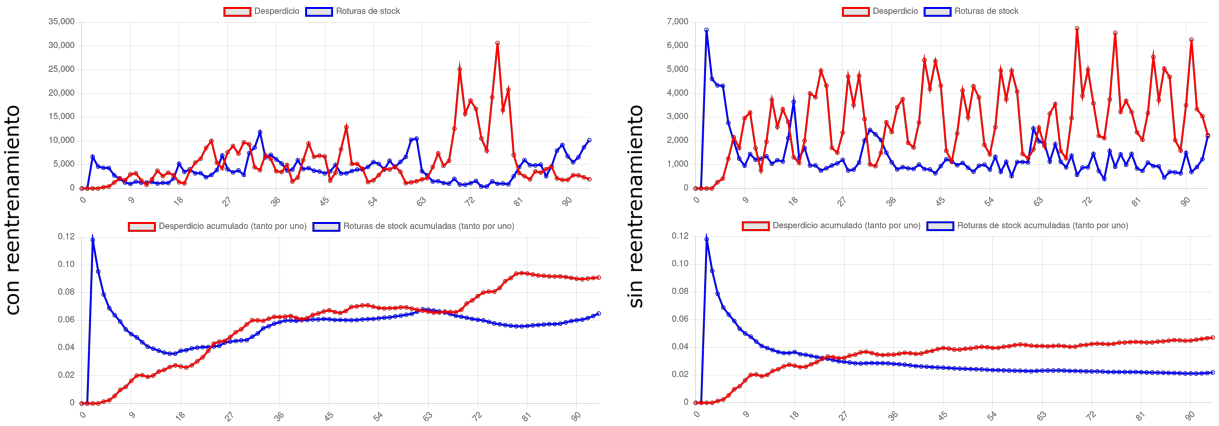


Figura 22: Gráficas de simulación para LSTM con reentrenamiento (izquierda) y sin él (derecha).

El reentrenamiento igualmente produce resultados bastante peores que sin reentrenamiento independientemente de los demás parámetros. En la figura 22 se ve cómo el reentrenamiento hace que se pase de un 6,9 % de suma de desperdicios y roturas de stock a un 15,6 %.

| | Calibrado | Sin calibrar | Sin CPM | Calibrado impreciso |
|---|---|---|---|---|
| Desperdicio | 0,5 % | 0,2 % | 0,4 % | 4,8 % |
| Roturas de stock | 2,4 % | 2,8 % | 3,1 % | 0,5 % |
| Total | 2,9 % | 3,0 % | 3,5 % | 5,3 % |

Cuadro 2: Resultados de ejecutar el sistema con y sin CPM, con y sin calibrar.

En el cuadro 2 podemos ver los resultados de las distintas opciones para el sistema con red bayesiana. En este caso, por limitaciones hardware, se han puesto 10 tiendas y 10 productos en los parámetros de la simulación.

Como es de esperar, el aprendizaje por refuerzo también mejora los resultados del sistema al hacer estimaciones teniendo en cuenta los niveles de stock del sistema (de forma, por ejemplo, que si hay más alimentos próximos a caducar en stock, prefiera tomar acciones por debajo de la predicción de demanda). Este algoritmo igualmente se ve beneficiado por más días de caducidad de los alimentos.

La calibración produce resultados levemente mejores: quizás esperábamos una mayor compensación que la que se ha producido entre desperdicios y roturas de stock con el modelo calibrado.

También podemos observar cómo un modelo calibrado pero poco preciso (se ha utilizado la RNP de la *técnica de modelado*) no obtiene buenos resultados. Un buen modelo debe poder encontrar un equilibrio entre la calibración y la precisión (*calibration vs sharpness*).

# 5

# Conclusiones y líneas futuras

## 5.1. Conclusiones

Lo primero que podemos sacar en claro de la realización de este proyecto es que la gestión de inventario no es una tarea sencilla. En este trabajo se ha realizado una versión muy simplificada de la gestión de inventario, obviando muchos problemas que aparecen en dicha gestión (como retrasos, devoluciones, robos, fallos a la hora de contabilizar la mercancía que entra y sale, etc.), tampoco se ha tenido en cuenta la parte económica de este proceso, y aún así, no ha sido fácil diseñar el sistema.

Se ha hecho una prueba: la de permitir el transporte entre tiendas de la misma ciudad cuando haya falta de stock; y no se han conseguido los resultados esperados. Y es que esta gestión es como un ecosistema, que cuando tocas algo, afecta a todo el sistema en modos que no siempre puedes prever.

En lo que respecta a la ciencia de datos, ha sido interesante comprobar las diferencias entre el aprendizaje supervisado y el aprendizaje por refuerzo, incluir ambos métodos en un mismo sistema y poder ver las diferencias tanto en resultados como a bajo nivel a la hora de depurar.

También ha resultado bastante flexible poder combinar el modelado de agentes con el aprendizaje automático. Nos ha permitido hacer de forma más fácil y ordenada distintos experimentos sobre el conjunto de datos. Así, por ejemplo, hemos podido cambiar fácilmente de algoritmo de aprendizaje, reducir o aumentar el número de tiendas o productos

según la simulación lo requiriese, hacer pruebas como la de reentrenamiento, y muchas más posibilidades que no han sido exploradas.

La elección de la plataforma Mesa ha sido muy acertada al permitirnos trabajar con un lenguaje muy propio para la ciencia de datos como es Python.

Por último, ha resultado gratificador trabajar con avances relativamente nuevos, como en este caso es la calibración para regresión, haciendo también que tengamos que trabajar con distribuciones de probabilidad en vez de valores deterministas, y viendo los resultados obtenidos con la estimación de incertidumbres.

## 5.2. Líneas futuras

En este trabajo nos hemos centrado únicamente en la gestión de inventario de productos perecederos.

La cadena de suministro de un supermercado es bastante amplia y compleja, e implica muchos procesos que deben coordinarse. Desde la coordinación con los proveedores de productos hasta la venta final, pasando por diferentes almacenes (almacenes regionales, locales y dentro de cada propio supermercado), así como distintos procesos como compras, devoluciones, robos, etc.

El módulo que hemos realizado se podría integrar en el desarrollo de un modelo basado en agentes de toda la cadena de suministro de un supermercado, de forma que se puedan estudiar y optimizar todos los procesos que ocurren en esta cadena.

Por otro lado, aquí se han utilizado algunos métodos concretos de AA. Otra línea de investigación sería la de utilizar y comparar una amplia gama de métodos de AA que se integren en nuestro modelo basado en agentes de gestión de inventario, y de esa forma saber qué clase de algoritmos se adaptan mejor a este tipo de problema.

Para poder seguir esta línea, se ha desarrollado la *Guía de programación*, donde se explica cómo implementar e integrar al sistema algoritmos de AA.

# Bibliografía


Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Descargado de https://www.tensorflow.org/ (Software disponible en tensorflow.org)

Agarap, A. F. M. (2018). Deep Learning using Rectified Linear Units (ReLU). *CoRR, abs/1803.08375*. Descargado 2022-07-20, de https://arxiv.org/pdf/1803.08375.pdf

Anthony, P., Soon, G., On, C., Alfred, R., y Lukose, D. (2014, 01). Agent Architecture: An Overview. *Transactions on Science and Technology*, 18–35. Descargado 2022-07-06, de https://www.researchgate.net/profile/Rayner_Alfred/publication/275643980_Agent_Architecture_An_Overview/links/555408a208ae980ca608703d/Agent-Architecture-An-Overview.pdf

Ben Fraj, M. (2017). *In Depth: Parameter tuning for Random Forest.* Descargado 2022-06-27, de https://medium.com/all-things-ai/in-depth-parameter-tuning-for-random-forest-d67bb7e920d

Cao, L. (2009). Introduction to Agent Mining Interaction and Integration. En L. Cao (Ed.), *Data Mining and Multi-agent Integration* (pp. 3–36). Boston, MA: Springer US. Descargado 2022-07-06, de https://doi.org/10.1007/978-1-4419-0522-2_1 doi: 10.1007/978-1-4419-0522-2_1

Corporación Favorita. (2018). *Corporación Favorita Grocery Sales Forecasting.* Kaggle. Descargado 2022-05-23, de https://www.kaggle.com/competitions/favorita-grocery-sales-forecasting/

Dudek, G., Jenkin, M. R. M., Milios, E., y Wilkes, D. (1993). Taxonomy for swarm robots. En *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93)* (Vol. 1, pp. 441–447). Piscataway, NJ. Descargado 2022-07-06, de https://www.researchgate.net/profile/David-Wilkes-2/publication/3687327_A_taxonomy_for_swarm_robots/links/00b7d52c83330639b4000000/A-taxonomy-for-swarm-robots.pdf?origin=publication_detail doi: 10.1109/IROS.1993.583135

Foundation for Intelligent Physical Agents. (2002). *FIPA Agent Management Specification.* Descargado 2022-07-06, de http://www.fipa.org/specs/fipa00023/SC00023J.html

Franklin, S., y Graesser, A. (1997). Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. En *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages* (pp. 21–35). Berlin, Heidelberg: Springer Berlin Heidelberg. Descargado 2022-07-05, de http://www.upv.es/sma/teoria/agentes/is%20it%20an%20agent-franklin.pdf

Gal, Y., y Ghahramani, Z. (2015). *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning.* arXiv. Descargado 2022-09-07, de https://arxiv.org/pdf/1506.02142.pdf doi: 10.48550/ARXIV.1506.02142

Heidenreich, H. (2018). *What are the types of machine learning?* Towards Data Science. Descargado 2022-07-17, de https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f

Ho, T. K. (1995). Random Decision Forests. En *Proceedings of 3rd International Conference on Document Analysis and Recognition* (Vol. 1, pp. 278–282). Descargado 2022-06-17, de https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=598994

Huhns, M., y Singh, M. (1998). *Agents and Multiagent Systems: Themes, Approaches and Challenges* (M. Huhns y M. Singh, Eds.). Morgan Kaufmann.

IBM Cloud Education. (2020). *Deep learning.* IBM Corporation. Descargado 2022-07-17, de https://www.ibm.com/cloud/learn/deep-learning

IBM Corporation. (2011). IBM SPSS Modeler CRISP-DM Guide [Manual de software informático]. Descargado 2022-05-26, de https://inseaddataanalytics.github.io/INSEADAnalytics/CRISP_DM.pdf

Jenkins, A. (2020). *Just-in-Time Inventory (JIT) Explained: A Guide.* Oracle Corporation. Descargado 2022-07-11, de https://www.netsuite.com/portal/resource/articles/inventory-management/just-in-time-inventory.shtml

Jensen, K. (2012). *CRISP-DM Process Diagram.* Wikimedia Commons. Descargado 2022-05-25, de https://commons.wikimedia.org/wiki/File:CRISP-DM_Process_Diagram.png

Kavlakoglu, E. (2020). *AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?* IBM Corporation. Descargado 2022-07-20, de https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks

Kazil, J., Masad, D., y Crooks, A. (2020). Utilizing Python for Agent-Based Modeling: The Mesa Framework. En R. Thomson, H. Bisgin, C. Dancy, A. Hyder, y M. Hussain (Eds.), *Social, Cultural, and Behavioral Modeling* (pp. 308–317). Cham: Springer International Publishing.

Kononenko, I. (1989). Bayesian neural networks. *Biological Cybernetics*, *61*(5),

361-370. Descargado de https://doi.org/10.1007/BF00200801 doi: 10.1007/BF00200801

Kuleshov, V., Fenner, N., y Ermon, S. (2018). Accurate Uncertainties for Deep Learning Using Calibrated Regression. *CoRR*, *abs/1807.00263*. Descargado 2022-08-13, de https://arxiv.org/pdf/1807.00263.pdf

Kuleshov, V., Seymour, H., Nemer, D., Meador, S., Fenner, N., y Schwartz, M. (2018). Towards a Sustainable Food Supply Chain Powered by Artificial Intelligence.. Descargado 2022-07-30, de https://www.cs.cornell.edu/~kuleshov/papers/icml2019-climate.pdf

Li, Y. (2017). Deep Reinforcement Learning: An Overview. *CoRR*, *abs/1701.07274*. Descargado 2022-07-20, de https://arxiv.org/pdf/1701.07274.pdf

Liaquat, I. (2020). *Linear Regression — Dummy Variable Trap.* Descargado 2022-06-22, de https://medium.com/analytics-vidhya/linear-regression-dummy-variable-trap-8964a83516d9

Malik, A., Kuleshov, V., Song, J., Nemer, D., Seymour, H., y Ermon, S. (2019). Calibrated Model-Based Deep Reinforcement Learning. En K. Chaudhuri y R. Salakhutdinov (Eds.), *Proceedings of the 36th International Conference on Machine Learning* (Vol. 97, pp. 4314–4323). PMLR. Descargado 2022-05-18, de https://proceedings.mlr.press/v97/malik19a/malik19a.pdf

Mitchell, T. (1997). *Machine learning.* McGraw Hill.

Peng, J., Jury, E., Dönnes, P., y Ciurtin, C. (2021). Machine Learning Techniques for Personalised Medicine Approaches in Immune-Mediated Chronic Inflammatory Diseases: Applications and Challenges. *Frontiers in Pharmacology*, *12*. Descargado 2022-07-12, de https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8514674/pdf/fphar-12-720694.pdf doi: 10.3389/fphar.2021.720694

Poslad, S., Buckle, P., y Hadingham, R. (2000). The FIPA-OS agent platform: Open source for open standards. *Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents*. Descargado 2022-07-06, de https://www.researchgate.net/profile/Stefan-Poslad/publication/228517710_The_FIPA-OS_agent_platform_Open_source_for_open_standards/links/0deec51b80dcbada60000000/The-FIPA-OS-agent-platform-Open-source-for-open-standards.pdf

Sutton, R. S., y Barto, A. G. (2017). Finite Markov Decision Processes. En *Reinforcement Learning: An Introduction* (pp. 47–71). Cambridge, MA: The MIT Press. Descargado 2022-07-20, de http://incompleteideas.net/sutton/book/RLbook2018.pdf

Thorn, J. (2020). *Random forest explained simply: An easy introduction to training, classification, and regression.* Towards Data Science. Descargado 2022-09-22, de

https://towardsdatascience.com/random-forest-explained-7eae084f3ebe

TIBCO Software Inc. (2021). *What is a Neural Network?* Descargado 2022-07-20, de https://www.tibco.com/reference-center/what-is-a-neural-network

United Nations Environment Programme. (2021). *Food Waste Index Report 2021.* Nairobi. Descargado 2022-05-25, de https://wedocs.unep.org/bitstream/handle/20.500.11822/35280/FoodWaste.pdf

Van Wyk, A. (2022). *Encoding Cyclical Features for Deep Learning.* Kaggle. Descargado 2022-09-22, de https://www.kaggle.com/code/avanwyk/encoding-cyclical-features-for-deep-learning/notebook

Wirth, R., y Hipp, J. (2000). *CRISP-DM: Towards a Standard Process Model for Data Mining.* Descargado 2022-06-01, de http://www.cs.unibo.it/~danilo.montesi/CBD/Beatriz/10.1.1.198.5133.pdf

Wooldridge, M. (2001). *An Introduction to MultiAgent Systems* (2.ª ed.). New York, NY: John Wiley & Sons, Inc.

Wooldridge, M., y Jennings, N. R. (1995). Agent Theories, Architectures, and Languages: A Survey. En M. J. Wooldridge y N. R. Jennings (Eds.), *Intelligent Agents* (pp. 1–22). Berlin, Heidelberg: Springer Berlin Heidelberg.

Work on Climate. (2021). *PAW Climate 2021 - Afresh: Towards a Sustainable Food Supply Chain Powered by AI.* Descargado 2022-09-07, de https://www.youtube.com/watch?v=hJKjNQFX6NY

Zappone, J. (2006). *Inventory Theory.* Whitman College. Descargado 2022-07-06, de https://www.whitman.edu/Documents/Academics/Mathematics/zapponj2.pdf

Zhang, W., Valencia, A., y Chang, N.-B. (2021). Synergistic Integration Between Machine Learning and Agent-Based Modeling: A Multidisciplinary Review. *IEEE Transactions on Neural Networks and Learning Systems*, 1–21. Descargado 2022-06-06, de https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9527397

# Apéndice A

# Manual de instalación

Este manual de instalación se ha desarrollado para que sea válido para cualquiera de los tres sistemas operativos predominantes para escritorio (Windows, MacOS y Linux) siempre que se cumplan los requerimientos detallados.

## A.1. Requerimientos

1. Al menos 16GB de memoria física y 30GB de memoria swap para las operaciones con el conjunto de datos.

2. Tener instalado Python en, al menos, la versión 3.8, y pip (el instalador de paquetes de Python).

3. Tener instalado R.

## A.2. Instalación

Llegados a este punto nos encontraremos con sistema con Python y R ya instalados, y con un archivo zip que corresponde a nuestro programa. Deberemos ejecutar los siguientes puntos para una instalación satisfactoria:

1. En primer lugar, descomprimir el archivo zip con nuestro programa en la carpeta en la que vayamos a trabajar con él.

2. Una vez descomprimido, hallaremos dos subcarpetas: una de ellas `Scripts` y la otra `InventoryManagement`. Entrar en la carpeta `Scripts` y abrir el archivo `kaggle-auth.txt`. En la primera línea sustituir *username* por el usuario de Kaggle, y en la segunda línea, *password* por la contraseña de Kaggle.

3. En la línea de comandos, ir a la carpeta `Scripts` y ejecutar el comando `Rscript kaggle.R`.

La ejecución del siguiente script puede tardar un tiempo considerable, dado que descargará los conjuntos de datos de Kaggle, los descomprimirá y los modificará para su posterior uso.

4. Una vez tenemos el conjunto de datos listo, falta por instalar las dependencias de Python.

   Para ello, nos iremos desde la línea de comandos a la carpeta `InventoryManagement` y ejecutaremos el siguiente comando: `pip install -r requirements.txt`. En caso de que convivan instalaciones de Python 2 y Python 3, habrá que usar la versión correcta de pip: `pip3 install -r requirements.txt`.

Ahora ya tendremos instalados todos los componentes necesarios para ejecutar nuestro software. Para ello, nuevamente desde la línea de comandos en la carpeta `InventoryManagement`, habrá que ejecutar el comando `python server.py`, que lanzará en una pestaña de un navegador web nuestro programa.

# Apéndice B

# Guía de uso

En esta sección se entiende que el usuario ha instalado correctamente el programa, tal y como se indica en el *manual de instalación*, incluyendo el comando para la ejecución del programa.

## B.1. Estructura

Cada vez que se lanza el software, el usuario suele tener que esperar unos segundos a que el programa cargue el conjunto de datos de entrenamiento (que es bastante grande), antes de que finalmente pueda ver la ventana del navegador.

Lo primero que se va a encontrar es la pestaña que se ve en la figura 23. En dicha pestaña tenemos los siguientes elementos:

- En la barra negra superior nos encontramos con el nombre de la aplicación, así como los botones *about* (que nos da una breve explicación de la función del programa), *start* (para comenzar la simulación), *step* (para ejecutar un solo paso de la simulación) y *reset* (que vale para resetear la simulación y además deberá ser pulsado cada vez que se cambie algún parámetro).

- En la parte izquierda de la ventana se pueden ver los parámetros que se pueden modificar de la simulación. Estos parámetros incluyen opciones tanto para el MSBA así como para los algoritmos de AA.

- En la parte central nos encontramos con tres elementos:

  - En primer lugar, una barra para subir o bajar la velocidad de simulación.
  - Justo debajo, la cuadrícula donde se visualiza la propia simulación.
  - Y por último, dos gráficas que irán mostrando la evolución. En la de arriba se puede observar el total de desperdicios y roturas de stock por cada día de la simulación; y en el de abajo, el porcentaje acumulado (en tanto por uno) de esos mismos dos parámetros.

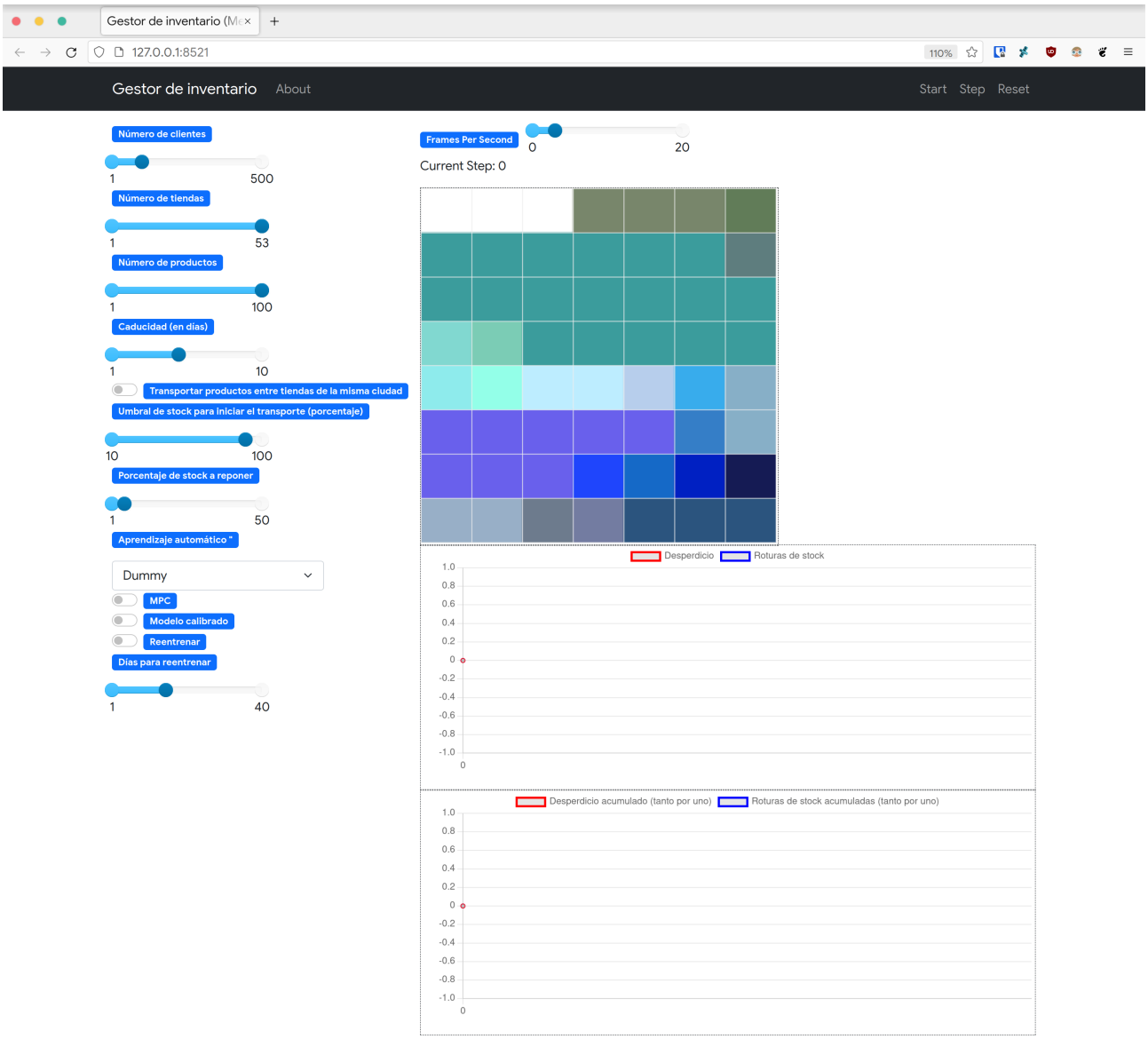


Figura 23: Estado inicial del software al ejecutarlo.

Para realizar una simulación, el usuario debe seleccionar los parámetros que vea convenientes, pulsar el botón *Reset*, esperar a que la cuadrícula se cargue, y entonces pulsar el botón *Start*.

## B.2. Parámetros

Se pueden distinguir dos tipos de parámetros: los que controlan de forma exclusiva la simulación, y los que afectan a cómo se comportan los algoritmos de AA.

Vamos a empezar describiendo el primer tipo de parámetros:

– *Número de clientes:* Dado que no tenemos datos sobre las transacciones por clientes que se realizan en las tiendas, nuestro sistema de MSBA simula el comportamiento

de los clientes generando un número de clientes que realizan compras aleatorias, sumando entre todos el número de compras por día que aparece en el conjunto de datos.

Muy útil para las simulaciones en las que se pueden transportar productos entre tiendas de la misma ciudad (ver más abajo). En caso de que esta opción no se active, se recomienda poner el número de clientes a 1 para tener una ejecución más fluida.

- *Número de tiendas:* El número de tiendas de la simulación. Se cogerán las n tiendas con mayor número de transacciones.

- *Número de productos:* El número de productos distintos que se utilizarán en la simulación. Se usarán los n productos más comprados.

- *Caducidad (en días):* Como en el conjunto de datos no aparece la caducidad de los productos, se establece un parámetro para todos ellos que marca cuántos días desde la llegada a la tienda tardan en caducar los productos.

- *Transportar productos entre tiendas de la misma ciudad:* Si se activa esta opción, cada vez que una tienda se encuentre falta de stock, puede pedir a las tiendas de la misma ciudad que tengan sobrante de stock los artículos que necesitan.

  Se recomienda tener un número de clientes elevado (al menos 100) en caso de que esta opción este activada, para una simulación más realista; y un número de clientes igual a 1 en caso de que esté desactivada (dado que el número de clientes no va a influir en la simulación y sí va a ralentizar la misma).

- *Umbral de stock para iniciar el transporte (porcentaje):* Si la opción de *transportar productos entre tiendas de la misma ciudad* está activada, cuando se llegue a este porcentaje de stock de alguno de sus productos la tienda entenderá que debe reponer de urgencia dicho producto (buscándolo en otras tiendas de la ciudad).

- *Porcentaje de stock a reponer:* Igualmente, si la opción de *transportar productos entre tiendas de la misma ciudad* está activada y se ha llegado al umbral de stock, la tienda intentará reponer stock en una cantidad igual al porcentaje indicado sobre el stock inicial.

Y ahora vamos a explicar los que tienen que ver con el AA:

- *Aprendizaje automático:* Se trata de una lista desplegable con las opciones de algoritmos de AA para la simulación.

Cada vez que se elija un algoritmo que no ha sido utilizado con anterioridad, el sistema tendrá una elevada espera antes de iniciar la simulación, puesto que deberá realizar el entrenamiento del algoritmo usando el conjunto de datos de entrenamiento.

La opción *Dummy*, que viene seleccionada por defecto, simplemente aplica un pequeño ruido sobre el valor correcto, y por tanto, no se trata de un algoritmo de aprendizaje automático. Su única función es la de comprobar que la parte de MSBA funciona correctamente, independientemente del algoritmo que se elija.

El resto de algoritmos no tienen en cuenta el valor correcto y devuelven una predicción sobre el estado actual. En la *guía de programación* se encuentra toda la información necesaria para añadir nuevos algoritmos.

- *CPM:* Si se activa esta opción se utilizará aprendizaje por refuerzo utilizando la técnica de control CPM.

- *Modelo calibrado:* Esta opción solo se tendrá en cuenta para sistemas bayesianos de AA, y no tendrá ninguna influencia en el resto de algoritmos. En caso de que esté activada, se calibrará la incertidumbre del sistema.

- *Reentrenar:* Si se quiere que se reentrene el regresor con datos ya simulados.

- *Días para reentrenar:* Cuando esté activada la opción de *reentrenar*, el número de días que tienen que pasar en la simulación para que se produzca un reentrenamiento.

Por último, queremos hacer hincapié en que cada vez que se haga algún cambio en los parámetros, hay que pulsar el botón de *reset* para que se contemplen dichos cambios en la simulación.

## B.3. Simulación

En este apartado se tratará de explicar el significado de los elementos visuales que aparecen en la cuadrícula de simulación.

Los cuadrados coloreados dentro de la cuadrícula corresponden a las diferentes tiendas de la simulación. Aquellas que tienen el mismo color indican que pertenecen a la misma ciudad, y por tanto, si se activa la opción, pueden transportar productos entre ellas.

Conforme se avance en la simulación, se verán unos círculos amarillos en cada cuadrado, como los de la figura 24. El radio de los círculos indica la suma de desperdicio y

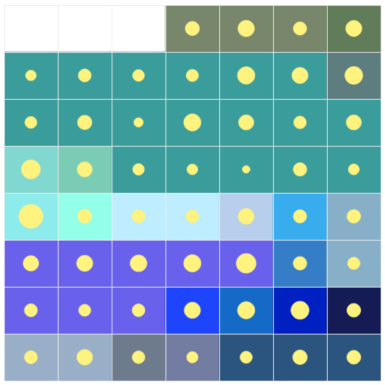

Figura 24: Círculos amarillos indicando la suma de desperdicio y roturas de stock en cada día y tienda.

roturas de stock en esa tienda y ese día de la simulación. Cuanto más grandes los círculos, mayor desperdicio y/o roturas de stock se habrán producido ese día en la tienda.

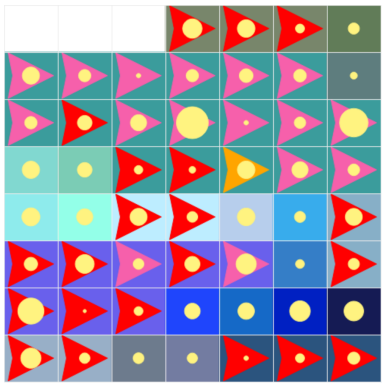

Figura 25: Flechas indicando el movimiento de stock entre tiendas. Las moradas indican que se han transportado productos desde esa tienda; las naranjas, que se han transportado a esa tienda; y las rojas, que ha habido movimientos desde y hacia esa tienda.

En caso de que se haya seleccionado la opción de *transportar productos entre tiendas de la misma ciudad*, además de los círculos amarillos, se aparecerán unas flechas, como se ve en la figura 25, que indican si se ha producido movimiento de stock entre tiendas. El color de las flechas en este caso indica lo siguiente:

- *Flechas moradas:* Indican que se han transportado productos desde esa tienda a otras que necesitaban el stock.

- *Flechas naranjas:* Indican que se han transportado productos a esa tienda por falta de stock.

- *Flechas rojas:* Indican que han salido productos a otras tiendas que los necesitaban, y han entrado otros productos (de otras clases) en los que había falta de stock.

La simulación termina cuando se simulan todas las fechas incluidas en el conjunto de prueba, y para entonces se habrán completado las gráficas con los resultados, tanto por día como acumulados de desperdicio y roturas de stock globales.

# Apéndice C

# Guía de programación

En esta guía se explica cómo implementar e integrar algoritmos de aprendizaje automático a nuestro sistema basado en agentes.

Estos algoritmos tendrán como función principal la de predecir el número de ventas por cada producto y por cada tienda que se producirán en un día concreto, de forma que el sistema pueda hacer una gestión de inventario eficiente.

En esta sección se asume que el usuario ha instalado el programa con las especificaciones dadas en el *manual de instalación*, y que tiene conocimiento del funcionamiento del mismo.

## C.1. Nomenclatura

Si exploramos la carpeta de instalación, vemos que dentro de la carpeta `Inventory-Management` se encuentra la subcarpeta `machine_learning`. Ahí es donde están ubicados los archivos Python que implementan los distintos algoritmos de AA.

De forma básica, para implementar un nuevo algoritmo, se debe crear en el directorio raíz `machine_learning` un archivo Python cuyo nombre debe ser el del algoritmo escrito usando *snake case*. Dicho archivo debe implementar una clase cuyo nombre debe ser el del algoritmo escrito usando *upper camel case*. Así, por ejemplo, si queremos implementar el algoritmo *linear regression*, debemos crear un archivo `linear_regression.py`, y dentro de ese archivo, implementar la clase `LinearRegression`.

En el caso de que se quisieran crear interfaces o clases abstractas, estas deben ir en un archivo Python que termine en `_interface.py` o `_abstract.py`. Estas terminaciones indican a nuestro sistema que no se tratan de algoritmos que deba mostrar en el desplegable. Las clases que se implementen en estos ficheros no tienen por qué seguir ninguna nomenclatura específica.

Si hubiese que crear clases que no fueran algoritmos de AA pero que se necesitaran para implementar dichos algoritmos, estas clases podrían ir en una subcarpeta de `machine_learning`, o en cualquier otra carpeta.

Es importante seguir la nomenclatura aquí descrita, puesto que si no, el sistema no será capaz de encontrar el algoritmo implementado, o fallará al intentar ejecutarlo.

## C.2. Regresor

Aunque no es estrictamente necesario, sí que es muy conveniente que los algoritmos que se implementen sean clases que hereden de la clase abstracta `Regressor`, donde se indican los métodos necesarios y se implementan algunos de los métodos comunes.

Pasamos a describir lo que se espera de los métodos abstractos que habría que implementar para heredar de esta clase:

- `_create_regressor(self)`:

  Crea y devuelve el modelo para el algoritmo de AA.

- `_train_regressor(self, x: DataFrame, y: DataFrame, x_cal: DataFrame = None, y_cal: DataFrame = None)`:

  Dado un conjunto de entrenamiento, un conjunto de etiquetas, un conjunto de calibración y otro conjunto de etiquetas de calibración, entrena y, si fuera necesario, calibra el regresor (el modelo que creamos con el método anterior).

  En caso de que no se tenga que calibrar el modelo, tanto `x_cal` como `y_cal` serán nulos.

- `_prediction(self, data: DataFrame) ->list`:

  Hace una predicción con el regresor sobre el conjunto de datos enviado, y devuelve la lista con las predicciones por cada dato.

- `_transform_x_parameters(self, df: DataFrame, has_labels: bool) ->DataFrame`:

  Hace las transformaciones necesarias del conjunto de datos a entrenar o predecir. Por ejemplo, las redes neuronales precisan de que los parámetros de entrada estén normalizados; así que este método haría la normalización necesaria.

- `_transform_y(self, df: DataFrame) ->DataFrame`:

  Hace las transformaciones necesarias del conjunto de etiquetas.

Con la implementación de esas clases sería suficiente para crear un algoritmo de AA que herede de la clase `Regressor` y pueda ser usado en la simulación.

## C.3. Redes neuronales

Para el caso de las redes neuronales, se ofrece ya implementada la clase abstracta `NeuralNetwork`, que implementa la mayor parte de las funciones necesarias para crear un regresor en Tensorflow (Abadi y cols., 2015).

Para heredar de esta clase solo hace falta implementar los siguientes métodos:

- `_create_model(self) ->Model`:

  Debe devolver un modelo secuencial o funcional de Tensorflow con la estructura de la red neuronal.

- `_transform_dataframe(self, df: DataFrame) ->DataFrame`:

  Devuelve el dataframe transformado. Este método es solo para el caso en el que haya que hacer transformaciones sobre la forma (`shape`) del dataframe, puesto que las transformaciones de normalización ya las implementa la propia clase.

También se encuentra la clase abstracta `BayesianNeuralNetwork` que hereda y tiene los mismos métodos abstractos que `NeuralNetwork`. Esta clase se utiliza para construir redes neuronales bayesianas que utilicen el sistema de *MC dropout* para obtener una aproximación gaussiana del modelo probabilístico.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA