Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

Una propuesta basada en aprendizaje automático para la mejora de la predicción de tiempos de llegada.

Alumno: Petar Georgiev Aleksandrov

Tutor/a/es: Miguel Ángel Martinez Prieto
Jorge Silvestre Vilches

Agradecimientos

En primer lugar quiero dar las gracias a mis tutores Miguel Ángel, Jorge y Aníbal por todo lo que me han ayudado a lo largo del desarrollo de este proyecto. También me gustaría agradecer a Boeing Research and Tecnology Europe por proporcionarme los datos que he usado en este estudio, sin estos no habría sido posible llevarlo a cabo. Además, me gustaría agradecer a todo el personal docente de la Escuela de Ingeniería Informática de Segovia su compromiso a la hora de impartir sus clases, si he llegado aquí es también gracias a lo que aprendí con todos ellos. Me gustaría dar las gracias a mis compañeros y personas cercanas por apoyarme en mis difíciles comienzos en el Grado, y por dedicarme tantas horas de su valioso tiempo. Y por ultimo quería agradecer y dedicar este trabajo a mi madre, gracias a su esfuerzo y sacrificio estoy hoy aquí.

"The greatest glory in living lies not in never falling,
but in rising every time we fall."
-Nelson Mandela
"Failure is an option here.If things are not failing,
you are not innovating enough."
-Elon Musk
"Pero cuando el loco se convirtió en genio
Resultó que todos sabían que lo iba a lograr"
-Natos y Waor

Resumen

Los últimos estudios realizados por Eurocontrol [30], indican que el 53 % de los vuelos en Europa sufrió algún tipo de retraso durante el año 2018. Aunque las causas de esta situación son variadas, la mayor congestión de los espacios aéreos es una de las más relevantes. Atendiendo a los informes de Eurocontrol elaborados a partir de estos estudios, se estima que el número de vuelos a nivel mundial, se ha incremento un 4,4 % en este último año respecto al año pasado, complicando aún más la gestión del tráfico aéreo (*ATM*).

Durante los últimos años se han desarrollado iniciativas de investigación centradas en modernizar y mejorar los sistemas ATM. Para ello, se están introduciendo nuevas tecnologías, que permitan aprovechar de una forma más efectiva los recursos de los que se dispone en el ámbito del tráfico aéreo, como por ejemplo ADS-B (*Automatic Dependent Surveillance Broadcast*). ADS-B es una tecnología de vigilancia que permite que la aeronave envíe información de su estado de forma periódica. Mediante estos mensajes seremos capaces de construir vectores que describen tanto la posición, como la altitud de la aeronave, así como otros parámetros del vuelo (velocidad horizontal y vertical, identificador de vuelo(*callsing*),...) en un determinado instante de tiempo. De esta forma, la trayectoria de un vuelo puede reconstruirse de acuerdo con sus vectores de estado y modelarse como una serie temporal. Sin embargo, el gran número de vuelos que se producen diariamente, junto con la gran cantidad de datos ADS-B que proporciona cada uno de ellos, hace que la explotación de toda esta cantidad de información tenga que abordarse utilizando tecnología Big Data.

En este Trabajo Fin de Grado (TFG) se abordará el reto de estimar los tiempos de llegada de los vuelos, en función de las trayectorias reconstruidas a partir ADS-B y de las variables que influyen en el trayecto de la aeronave. Toda esta información será, posteriormente, utilizada para analizar los retrasos sufridos por estos vuelos, proponiendo nuevos mecanismos para mejorar la estimación actual de los tiempos de llegada. Para realizar las predicciones hemos usado redes LSTM que permiten basar su predicción en mensajes anteriores es una técnica muy usada en problemas que tienen como base las series temporales.

Finalmente, cabe destacar que este TFG se desarrollará de forma iterativa e incremental bajo la metodología UVAGILE, permitiendo dar un enfoque más dinámico al proyecto y posibilitando que su resultado final se ajuste mas y mejor a los objetivos propuestos al comienzo del proyecto.

Palabras claves: Aprendizaje automático, Predicción, Gestión de trafico aéreo, tiempos de llegada, ADS-B, LSTM.

Abstract

The latest studies carried out by Eurocontrol [30], indicate that 53 % of flights in Europe during 2018 suffered some sort of delay. Although the causes of this situation are varied, increased airspace congestion is one of the most significant. According to Eurocontrol reports, based on these studies, it is estimated that the number of flights worldwide has increased by 4.4 % in the last year, further complicating air traffic management (ATM).

In recent years, research initiatives have been developed that focus on modernising and improving ATM systems. To this end, new technologies are being introduced to make more effective use of the resources available in the field of air traffic, such as ADS-B (Automatic Dependent Surveillance Broadcast). ADS-B is a surveillance technology that allows the aircraft to send status information on a regular basis. Through these messages we are able to build vectors that describe both the position and the altitude of the aircraft, as well as other flight parameters (horizontal and vertical speed, callsing) at a given instant of time. In this way, the trajectory of a flight can be reconstructed according to its state vectors and modeled as a time series. However, the large number of flights that occur daily, together with the large amount of ADS-B data that each one provides, means that the exploitation of all this amount of information has to be addressed using Big Data

This work will address the challenge of estimating flight arrival times (at their destination), based on the trajectories reconstructed from ADS-B and the variables that influence the path of the aircraft. All this information will later be used to analyse the delays suffered by these flights, proposing new mechanisms to improve the current estimate of arrival. To make the predictions we have used LSTM networks that allow to base their prediction in previous messages is a technique very used in problems that have as base the time series.

Finally, it should be noted that this thesis will be developed in an iterative and incremental way under the methodology UVAGILE, allowing to give a more dynamic approach to the project and enabling that its final result fits more and better to the objectives proposed at the beginning of the project.

Key words: Machine Learning, Forecasting, Air Traffic Management, Arrival Times, ADS-B, LSTM.

Índice general

Índice de figuras

Capítulo 1

Introducción

Actualmente nos encontramos en pleno auge de la era digital. El Big Data / Inteligencia Artificial han sido el motor de esta nueva era digital, la cual ha permitido optimizar y mejorar casi cualquier aspecto de nuestra vida, desde algo tan sencillo como sugerir algún producto en función de otro que hemos adquirido, a algo tan complejo como es la conducción autónoma. Estos cambios se reflejan en un amplio abanico de sectores dentro de nuestra sociedad, como la medicina, enseñanza, sociología, etc...

En campos como la aviación ha permitido grandes avances, como los aviones no tripulados, la optimización de rutas en función de la meteorología o la optimización de tiempos para mejorar el manejo de los recursos, ya que en este sector los recursos son muy costosos y difíciles de manejar. Estas tecnologías han tenido una gran evolución desde su aparición y no paran de crecer a día de hoy, lo que motiva una gran demanada de profesionales especializados en esta rama de la ingeniería informática.

1.1. Motivación

En los últimos años el creciente número de vuelos ha supuesto un gran problema para la gestión del tráfico aéreo. La cantidad de vuelos a coordinar provoca retrasos en las salidas y llegadas de los vuelos, cancelaciones, dificultad para prever el personal necesario en el aeropuerto para hacerse cargo de las llegadas, etc...

Todo esto hace que sea muy difícil gestionar los recursos de manera eficiente: según Eurocontrol en 2018 el coste de los retrasos en vuelos ha supuesto 17.600 millones de euros [25]. Y a esto se suma que en el año 2018, según otro estudio llevado por Eurocontrol, hubo un aumento de los retrasos en vuelos del 53 %, y los pronósticos para el 2019 no fueron mejores, ya que se preveía que estos factores seguirán aumentando [30]. Estos retrasos afectan significativamente a la ruta del avión, haciendo que su trayecto pueda extenderse y esto suponga un coste añadido por cada vuelo que sufre retraso. Costes provenientes del despliegue de todos los operarios que tienen que atender al avión en el momento de llegada fuera del momento previsto. Y por este motivo, se quiere mejorar la gestión del tráfico aéreo.

La gestión del tráfico aéreo (ATM, *Air Traffic Management*) es el conjunto de sistemas que se encargan de la asistencia a las aeronaves desde el aeropuerto de origen hasta la llegada al aeropuerto de destino, incluyendo todo el trayecto entre ambos aeropuertos [20]. En ese trayecto, que se denomina espacio aéreo de tránsito, el avión será dirigido por diferentes torres de control en función de dónde se encuentre. Por tanto, un avión a lo largo de su vuelo atravesará diferentes sectores aéreos, por los cuales será guiado por diferentes torres de control. Por consiguiente, como podemos imaginar, operar con tantos vuelos es una tarea compleja a la que hay que añadirle factores como la meteorología o el grado de saturación del espacio aéreo en ese momento.

Analizando las previsiones sobre el ámbito ATM, estas indican que tanto en cantidad como en complejidad seguirán una trayectoria creciente y por esto, es necesario desarrollar nuevas técnicas con el fin de adaptarse a esta situación. En la actualidad no se cuenta con técnicas suficientes para satisfacer estos problemas y contemplar todas las variables que intervienen en una correcta gestión del trafico aéreo.

Figura 1.1: Tráfico Aéreo Mundial en tiempo real [18].

Para esta finalidad, es imprescindible contar con datos reales y de buena calidad con el fin de desarrollar y evaluar estas nuevas soluciones. En la actualidad existen iniciativas como es Flightradar24 [1], que da información sobre los vuelos en tiempo real, como puede verse en la Figura 1.1. Su principal fuente de datos son los dispositivos ADS-B (*Automatic Dependent Surveillance - Broadcast*), que definiremos más adelante, y que proporcionan

[1] https://www.flightradar24.com/

datos sobre la ruta e información de la aeronave. Según MBA [15] la profesión de controlador aéreo es de las mas estresantes del mundo, esto es debido a la gran cantidad de vuelos con los que deben operar cada día.

El estado de los vuelos a nivel europeo, y su problemática, son similares a los que hay a nivel global. Se experimentan retrasos a causa de la dificultad de la gestión del tráfico aéreo, lo que provoca sobrecostes en combustible y en personal, tanto en la parte de tripulación aérea como en la parte de personal del aeropuerto. Sin embargo, Europa tiene algunos de los aeropuertos mas saturados a nivel mundial, y por tanto es más susceptible a retrasos y a generar el sobrecoste del que hablábamos. En la Figura 1.2 podemos ver una tabla con los 25 aeropuertos con más tráfico del mundo [39].

Rank	Airport	Country	Total passengers	Rank change
1.	Hartsfield–Jackson Atlanta International Airport	United States	110,531,300	—
2.	Beijing Capital International Airport	China	100,011,000	—
3.	Los Angeles International Airport	United States	88,068,013	▲1
4.	Dubai International Airport	United Arab Emirates	86,396,757	▼1
5.	Tokyo Haneda Airport	Japan	85,505,054	—
6.	O'Hare International Airport	United States	84,397,776	—
7.	London Heathrow Airport	United Kingdom	80,844,310	—
8.	Shanghai Pudong International Airport	China	76,153,500	▲1
9.	Charles de Gaulle Airport	France	76,150,007	▲1
10.	Dallas/Fort Worth International Airport	United States	75,066,956	▲5
11.	Guangzhou Baiyun International Airport	China	73,378,475	▲2
12.	Amsterdam Airport Schiphol	Netherlands	71,706,999	▼1
13.	Hong Kong International Airport	Hong Kong SAR, China	71,541,000	▼5
14.	Frankfurt Airport	Germany	70,560,987	—
15.	Denver International Airport	United States	69,015,703	▲5
16.	Indira Gandhi International Airport	India	68,490,731	▼4
17.	Seoul Incheon International Airport	South Korea	68,350,784	▼1
18.	Singapore Changi Airport	Singapore	68,300,000	▲1
19.	Suvarnabhumi Airport	Thailand	65,424,697	▲2
20.	Soekarno–Hatta International Airport	Indonesia		▼2
21.	John F. Kennedy International Airport	United States	62,551,072	▲1
22.	Kuala Lumpur International Airport	Malaysia		▲1
23.	Madrid Barajas Airport	Spain	61,734,037	▲1
24.	San Francisco International Airport	United States	57,488,023	▲1
25.	Chengdu Shuangliu International Airport	China	55,858,552	▲1

Figura 1.2: Top 25 aeropuertos con más tráfico del mundo.

Como podemos apreciar, de 25 aeropuertos, 5 son aeropuertos europeos y el resto están repartidos por diversos continentes. Según Airlines [31] más de 210.000 vuelos en Europa

han sufrido retrasos en el mes de Junio del año 2019, prácticamente 20 % del total de vuelos en ese mes, el retraso medio en cada vuelo fue de unos 17 minutos.

A raíz de lo visto anteriormente, han surgido diferentes iniciativas que pretenden dar solución a estos problemas de retrasos o por lo menos mejorar los existentes. Surgen iniciativas como el Cielo Único Europeo (*Single European Sky*, SES) [11]: es una propuesta de la Unión Europea, a través de Eurocontrol, con la que se pretende unificar la gestión de la navegación aérea de Europa con el fin de hacer un transporte más eficiente y evitar costes añadidos. Esta propuesta consta de varios programas como SESAR (*Single European Sky ATM Research*) que pretende la modernización de la gestión del tráfico aéreo. Pretenden continuar por lo menos hasta el año 2024 con la fase de I+D. Una de las principales líneas de investigación sobre las que se esta trabajando es la gestión de trayectorias 4D, que deben su nombre a las dimensiones de altitud, latitud, longitud y tiempo. Esta última es la variable que nos habilita para realizar una predicción que permita prever en qué momento llegará el avión al aeropuerto, con diferentes rangos de antelación, y sobre todo poder realizar rutas más óptimas con la menor distorsión posible. Esta propuesta hace acopio de las nuevas tecnologías para la mejora de las soluciones actuales al problema expuesto anteriormente: una de las más relevantes en esta transformación es la integración y uso de sistemas ADS-B tecnología que nos brinda información sobre el posicionamiento en tiempo real de las aeronaves con una precisión aún más alta que los radares anteriormente usados, destacando además que el costo de esta tecnología es más bajo. Lo que se pretende con todas estas propuestas es hacer el vuelo lo más directo posible sin que este tenga que hacer desvíos imprevistos en su trayectoria y que así se cumpla la predicción del tiempo de llegada para que todos los medios estén disponibles y la aeronave gaste el menor combustible extra posible.

1.2. Objetivos y restricciones

En este trabajo de fin de grado, hemos identificado 3 objetivos principales que a su vez hemos dividido en sub-objetivos, con el fin de facilitar la consecución de los objetivos principales.

Objetivos del proyecto:

- **OBJ-1** Realizar un estudio sobre la predicción de tiempos de llegada de aeronaves.
  - OBJ-1.1 Estudio sobre el contexto del problema de la gestión aérea.
  - OBJ-1.2 Estudio sobre series temporales y su aplicación a problemas de aprendizaje automático.
  - OBJ-1.3 Estudio sobre el uso de bases de datos especializadas en series temporales y su importancia aplicado a nuestro problema.
- **OBJ-2** Desarrollo de un modelo predictivo adaptado a la operativa del aeropuerto de Madrid Barajas Adolfo Suárez.

  - OBJ-2.1 Datos procesados para la ingesta del modelo.
  - OBJ-2.2 Creación de modelo LSTM basado en redes neuronales.
- **OBJ-3** Construcción de un cuadro de mandos interactivo que nos permita visualizar y comprender los datos con mayor facilidad.
  - OBJ-3.1 Adecuación de funciones para Dashboard.
  - OBJ-3.2 Ingesta de datos provenientes de nuestros archivos.
  - OBJ-3.3 Creación de gráficos.
  - OBJ-3.4 Gráficos capaces de interaccionar entre ellos.
  - OBJ-3.5 Despliegue de aplicación.

Restricciones El desarrollo de este proyecto conlleva una serie de limitaciones, entre las cuales podemos encontrar:

- Disponibilidad de tiempo para el desarrollo del proyecto, al ser un proyecto limitado a 300 horas. Por esta razón, debemos tener cuidado a la hora de definir el alcance de nuestro proyecto al tratarse de un problema complejo y que tiene cierto grado de investigación.
- Limitación en cuanto a volumen de datos, ya que disponemos de datos recogidos durante un solo día. Esto puede ocasionar que no obtengamos la máxima precisión a la hora de predecir, ya que no podemos explotar todo el potencial de las LSTM para explotar la estacionalidad de los datos o la presencia de patrones, por ejemplo en fechas concretas, a lo largo de un año o de los días de la semana.
- Rutas transatlánticas, al no disponer de receptores ADS-B capaces de recibir mensajes sobre el océano Atlántico, no disponemos de datos en esa parte del trayecto, por tanto, no podemos realizar predicciones certeras para algunos vuelos transatlánticos que sobrevuelan la zona, ya que no contamos con suficientes mensajes.
- Privacidad de los datos, los datos son cedidos por Boeing Research & Technology Europre (BRT-E) dentro de un proyecto de colaboración con la Universidad de Valladolid (UVa) por tanto no podremos realizar un posterior despliegue a un servicio externo en la nube.

1.3. Estructura del documento

- **Capítulo 1. Introducción.** Capítulo introductorio del proyecto, en el se exponen los objetivos y la motivación para llevar a cabo este proyecto.
- **Capítulo 2. Gestión del Tráfico Aéreo.** En este capítulo se describe el entorno de la gestión del trafico aéreo, así como sus diferentes fases a lo largo de un vuelo.

- **Capítulo 3. Gestión del Proyecto.** Capítulo en el cual describiremos la metodología adoptada para la realización del proyecto, así como su presupuestación y balance.
- **Capítulo 4. Series Temporales.** En este capítulo realizaremos un estudio de los modelos existentes aplicados a la estimación de tiempos.
- **Capítulo 5. Bases de Datos de Series Temporales.** Capítulo dedicado al estudio de las bases de datos de series temporales, incluyendo su análisis, evaluación y comparativas.
- **Capítulo 6. Construcción del Modelo de Aprendizaje.** En este capítulo, definiremos el proceso llevado a cabo para la creación del modelo de aprendizaje capaz de realizar las predicciones.
- **Capítulo 7. Resultados.** Capítulo dedicado al análisis de los resultados obtenidos a lo largo del estudio.
- **Capítulo 8. Construcción del Dashboard.** En este capítulo describiremos cómo se ha realizado la construcción de la aplicación web, así como las herramientas que se han necesitado para la implementación y las tecnologías usadas.
- **Capítulo 9. Conclusiones y Trabajo Futuro.** Último capítulo de la memoria, en él se hará una reflexión sobre el trabajo realizado así como futuras mejoras a realizar.

Capítulo 2

Gestión del tráfico aéreo

En este capítulo haremos un estudio del entorno de nuestro problema, con el fin de entender mejor cómo funciona la gestión del tráfico aéreo. Esto nos ayudará a comprender los factores que influyen a lo largo de las diferentes fases de vuelo.

2.1. Introducción

La gestión del tráfico aéreo, o como es conocida en inglés, ATM (*Air Traffic Management*), se define como todos los mecanismos necesarios para que un avión sea capaz de despegar, y poder acceder a un espacio aéreo transitado por más aviones, con el fin de posteriormente tomar tierra en otro aeropuerto destino [20]. Entre estos mecanismos podemos encontrarnos: Control del Tránsito Aéreo (ATC), Gestión de Afluencia y Capacidad (ATFCM), los Servicios de Control del Flujo (ATS), Servicios de Gestión del Espacio (ASM), sistemas encargados de la navegación, la meteorología aeronáutica y la seguridad del tránsito aéreo.

En la gestión del tráfico aéreo existen varias zonas bien diferenciadas, como podemos ver en la (Figura 2.1), que pueden abarcar diferentes espacios aéreos. Se pretende que estos sistemas sean muy homogéneos y que permitan a la aeronave desplazarse sin que sea necesario un cambio brusco de condiciones de vuelo, y así poder transitar de un espacio aéreo a otro.

Figura 2.1: Etapas vuelo.

Con el fin de entender mejor el funcionamiento de un vuelo, lo separaremos en 7 fases [20]:

1. **Introducción**: el espacio aéreo está separado por una serie de segmentos invisibles, cada uno de estos gestionado por su propio ATC. Cada ATC debe conocer los datos generales de los vuelos en ese área de control, también tiene que estar conectado con las áreas contiguas para facilitar la transición del vuelo de un segmento a otro. Los segmentos deben estar comunicados entre sí para que cuando una aeronave pase de un segmento a otro la comunicación perdure y no haya incidentes.

2. **Antes del despegue**: en este apartado se revisan varios documentos importantes. Uno de ellos es el plan de vuelo, también conocido como OFP (*Official Flight Plan*) el cual es enviado a todos los ATC's por los que el avión pasa. En el caso de los vuelos Europeos el plan de vuelo es mandado a IFPS (*Integrated Flight Plan Processing System*). Una vez IFPS recibe el plan de vuelo, se encarga de de difundirlo a la torre de control y todos los organismos implicados en la gestión del vuelo. No para todos los vuelos es necesario un plan de vuelo, pero en estos casos sí es necesario:

   - Vuelos IFR (*Intrument Flight Rules*). Casi todos los vuelos de origen comercial son IFR. Esto implica que no es necesario tener contacto visual con el terreno ya que disponen de sistemas de navegación.
   - Los que atraviesan fronteras internacionales.
   - Los que vuelan bajo el estandarte IFR en un espacio aéreo definido, el cual proporciona información sobre otros vuelos en caso de ser necesario.
   - Si este requiere servicio de ATC (*Air Traffic Control*).
   - Que el vuelo pase por áreas que requieran uso del ATS (*Air Traffic Service*), que les proporciona información del vuelo, búsqueda y rescate de ser necesarios.

Los planes de vuelo no se realizan varias veces si el trayecto es el mismo, de hecho se usan RPL (*Repetitive Flight Plan*) que consiste en reutilizar un plan previo. Por tanto, esto permite facilitar las cosas y que haya que realizar entre uno y dos planes de vuelo al año por aerolínea [20]. En todo caso el piloto debe ser informado si existe algún cambio dentro del ATM en caso de afectar la ruta del vuelo.

Otro documento importante que se debe emplear es el NOTAM (*Notification to Airmen*): consiste en avisos que se presentan a las autoridades de aviación para alertar a los pilotos de algún peligro a lo largo de la ruta. Las razones más comunes por las que se manda un NOTAM son:

- Peligros en exhibiciones aéreas o saltos de paracaídas.
- Activación o desactivación de zonas restringidas.
- Presencia de obstáculos en el vuelo.
- Ejercicios militares que imponen restricciones en el espacio aéreo.

En el documento sobre el clima conocido también como WX (*Weather Document*), podemos encontrar varios datos:

- TAF (*Terminal Aerodrome Forecast*) aporta información meteorológica del aeropuerto destino y uno alternativo en caso de emergencias.
- Pronóstico a lo largo de la ruta de vuelo.
- Información de viento y temperatura para rango de altitud baja del vuelo.
- Información de viento y temperatura a altitud de crucero es la franja entre los 10.500 y los 12.000 metros de altura.

El ATC es informado también sobre el estado atmosférico, ya que es un factor crucial que deben tener en cuenta en todo momento, destacando entre estos datos el de dirección del viento y velocidad del viento en la pista.

3. **Despegue**: el despegue es una de las funciones principales de las que se encarga la torre de control. Con el fin de asegurar una correcta ejecución, debe tener bajo control los siguientes factores:

   - Taxi: En este estado se debe controlar y planificar el movimiento de la aeronave en tierra.
   - Despegue: Debe controlar las diferentes salidas del aeropuerto.
   - Aterrizaje: Control de aeronave que aterriza desde que se aproxima hasta que toca tierra por primera vez.
   - Vuelos visualmente cercanos: esto se da en aeropuertos de tamaño más pequeño. El controlador puede ver el avión, y proceder a dar las indicaciones. Pero en aeropuertos de tamaño más grande en el cual no puedes ver el avión a simple vista, se usa control de aproximación o control de terminal.

El ATC es la autoridad que proporciona información tanto al aeropuerto como al avión sobre el tiempo, pistas en uso y otros datos... También está el ATIS (*Aerodrome Terminal Information Service*) cuya labor es proporcionar información general del aeropuerto como: datos meteorológicos, condiciones en las que se deben operar, pistas en uso, etc...

Los controladores en las torres de control disponen de los detalles del vuelo proporcionados por IFPS, y también cuentan con segmentos de ATC y detalles proporcionados por el plan de vuelo, esta información se proporciona en forma de tiras (ver Figura 2.2) y son actualizadas por el controlador.

Figura 2.2: Tiras plan vuelo.

A cada vuelo se le reserva una franja horaria en la que puedan operar, esto es gracias al CTOT (*Calculated Take-Off Time*) o Franja de tiempo (*SLOT*) es la hora estimada de despegue de la aeronave, es una franja horaria reservada, en la que el vuelo tiene preferencia con el fin de poder cumplir con el horario previsto en el Plan de vuelo. En este intervienen los CFMU (*Control Flow Management*) que controlan los tiempos medios de rotación para los campos de vuelo o pistas, y que se responsabilizan directamente de regular las salidas para que ATM no sufra sobrecarga. Todo esto se calcula en base a las condiciones atmosféricas y restricciones temporales que sufre el espacio aéreo.

CFMU es una extensión que sirve de complemento al ATC con el fin de equilibrar la demanda de capacidad y mantener los retrasos y la congestión área al mínimo dentro del espacio aéreo Europeo. El asistente (el ordenador) recibe la tira anteriormente citada, 40 minutos antes que el controlador reciba el CFMU. Una vez el CFMU es aceptado, es el asistente el encargado de escribir el SLOT junto la ruta que debe tomar la aeronave después del despegue. Otra función del asistente será la actualización en el sistema informático para que las otras ATC conozcan el CFMU aceptado.

El piloto recibe CFMU de la compañía, y en ese momento la tira pasa del controlador al plan de movimiento en tierra. Estos aprueban el inicio y dan permiso al avión para que se dirija a la pista correspondiente. La autorización incluye el código squawk, un código de 4 cifras que oscilan entre el 0 y 7, y que el piloto debe de introducir en el transpondedor de la aeronave para que así el avión pueda ser identificado por los radares ATC. Esto también incluye instrucciones de la ruta que seguirá la aeronave durante la primera franja, que se delimita desde el despegue hasta unos

instantes posteriores al despegue. Estos constituyen una salida estándar dentro de la instrumentación SID (*Standard instrument departure*) donde se publican los procedimientos que deben seguir las aeronaves. Los SID son necesarios para llevar el avión desde la pista hasta el primer punto de ruta denominado *waypoint*.

Cuando las instrucciones son recibidas por la torre de control, es el piloto el que debe repetir las instrucciones recibidas (*read-back*), podemos decir que este tiene un sentido cíclico como podemos ver en la (Figura 2.3):

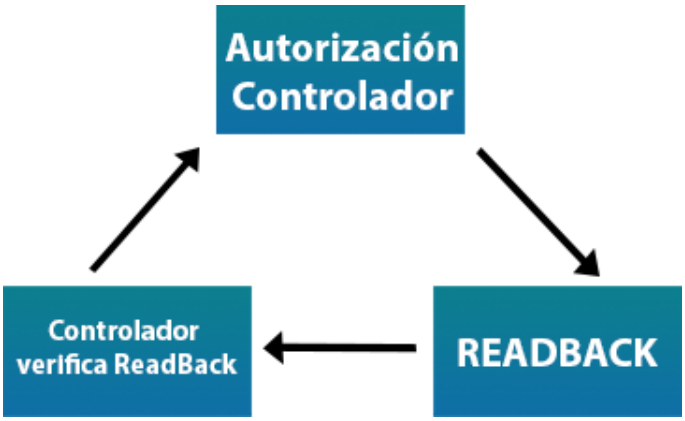


Figura 2.3: Ciclo Autorización.

4. **Salida**: este apartado corresponde a cuando el avión está en el aire. Es el controlador de salidas el encargado de guiar el avión hasta que este llegue a cierto punto, en el cual se encargará otro controlador. El sistema informático se encarga de calcular el tiempo necesario para que el avión alcance el primer punto de cada segmento aéreo para pasar de un control a otro, basándose sobre todo en datos conocidos de maniobras de aeronave y los pronósticos de viento.

   El plan de vuelo se activa cuando el radar identifica la aeronave y empieza a realizar un seguimiento continuo de esta. En este instante los controladores muestran las tiras actuales en todos los segmentos de la ruta usando la hora de despegue real. En ese momento el controlador se encargará de dar instrucciones y autorizar maniobras para asegurar la seguridad y eficiencia, si surge alguna incidencia a lo largo del trayecto se anotará en la tira.

   Cuando el avión alcance el punto anteriormente comentado, se deberán comunicar entre ambos segmentos con el fin de coordinar el traspaso de la aeronave de un segmento a otro.

5. **En vuelo**: mientras el avión esté viajando cruzará varios segmentos de ATC y estos a su vez pueden subdividirse en varios sectores, por tanto, debemos saber que cada sector puede estar operado por uno o dos controladores, cada uno de estos con una capacidad aproximada para manejar hasta 15 aviones. Existen dos tipos de control:

- Procesal: Esta es usada como apoyo. Principalmente se basa en informes de la posición y estimaciones. Se usan las tiras para poder planear rutas seguras.
- Radar: Este permite reducir el numero de tiras requerido y nos permite operar con separaciones mínimas entre aeronaves.

El piloto debe prestar atención a Redfa: este es un punto del informe ATC, está definido por 5 letras y significa que en el punto en el que se encuentra no hay radio ayudas. Cualquier otro cambio o revisión debería ser comunicado vía telefónica.

Cuando la aeronave está entrando en un segmento diferente aparece en el radar parpadeando y es el controlador el que debe aceptarla o rechazarla. Para ello el radar muestra la siguiente información:

- Un símbolo que aparece junto a una etiqueta que identifica el vuelo, que proporciona datos como el estado, o el nivel de vuelo planeado.
- Puntos de ruta FIX: tienen forma triangular, se asocian a balizas que ayudan a la navegación en tierra, esto permite al piloto informar de su posición relativa a un FIX.

6. **Descenso**: antes de que el avión comience las maniobras de descenso, el piloto tiene que validar la información de destino. Esa información es la siguiente:

   - VOLMET es una red global de estaciones de radio que transmiten datos meteorológicos a través de radio de onda corta.
   - ATIS es el servicio de información terminal del aeropuerto, este nos proporciona información general del aeropuerto.

   Una vez realizadas estas consultas, desde la torre de control le dan permiso al avión para empezar a realizar maniobras de aproximación, este es orquestado por el Control de aproximación. En caso de encontrarnos con mucho tráfico el avión sera puesto en HOLD, esto hará que el avión haga maniobras de espera hasta que le autoricen el descenso. No en todos los aeropuertos existen las denominadas STAR, que son rutas de aproximación predefinidas, que se deben pasar de manera obligatoria antes de dar el visto bueno para el descenso, con el objetivo de agilizar y aumentar la eficiencia con la que se realizan estas maniobras. Una vez esté todo correcto, el controlador proporciona la pista de aterrizaje, y el tipo de aproximación puede ser por Radio frecuencia (VOR) o por sistema de aterrizaje instrumental (ILS).

7. **Aterrizaje**: una vez el avión esté en tierra, el piloto deberá contactar con torre de control para seguir el procedimiento de aterrizaje pero a la inversa, hasta llegar a una manga del aeropuerto. Una vez finalizado el vuelo, todos los datos del ordenador y tiras quedan almacenadas en un registro.

2.2. ADS-B.

ADS-B es una tecnología de vigilancia del espacio aéreo que proporciona a las aeronaves una vía para transmitir datos recogidos por sus sistemas de navegación. Estos dispositivos se caracterizan por tres características principales [19].

1. Precisión: La señal de posicionamiento global (GPS) que nos facilitan los sistemas ADS-B es mucho más precisa que las que se obtienen por radar. Esto es muy importante para la mejora de la gestión del espacio aéreo, ya que podríamos dejar menos separación entre las aeronaves, al conocer con más precisión donde están. Esto es una medida excelente ante la problemática del espacio aéreo congestionado.

2. Simplicidad: Los dispositivos de vigilancia ADS-B nos brindan una mayor simplicidad operativa y por tanto la infraestructura tiene un menor coste que la de un radar convencional de tierra.

3. Multiuso: Es un servicio de difusión (*broadcast*), por tanto, las otras aeronaves y los servicios de control aéreo pueden tener acceso a información precisa de la presencia de otras aeronaves en las inmediaciones.

Para explicar el funcionamiento del ADS-B vamos a hacer uso de la Figura 2.4 con el fin de esclarecer en la mayor medida posible cómo funciona.

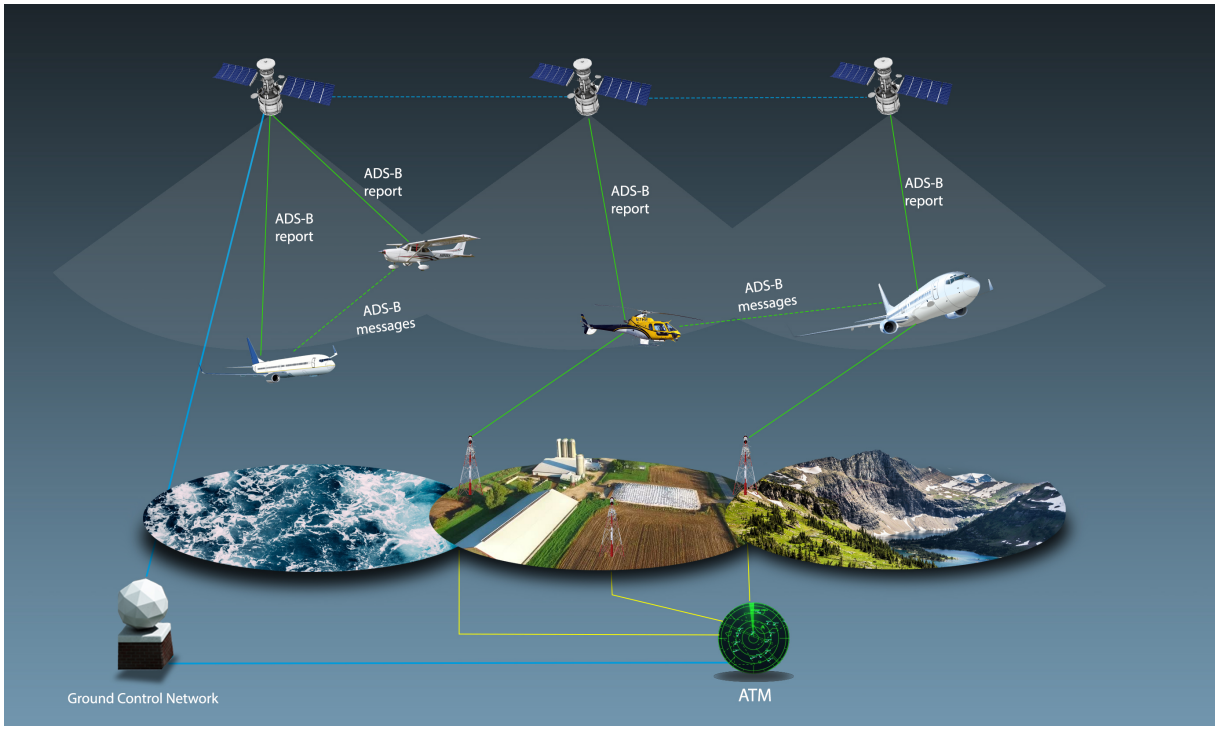


Figura 2.4: Esquema comunicación ADS-B.

El funcionamiento de esta tecnología consiste en que las aeronaves recopilan datos haciendo uso de sus sensores, además de los datos de su posición que obtienen gracias a los satélites. Una vez tiene estos datos, de manera automática son enviados mediante *broadcast* (difusión) y todos aquellos posibles receptores que estén dentro de su rango de acción capturan esta información (tanto las otras aeronaves dentro del espacio aéreo, como controladores, y todos los dispositivos preparados para recibir estos mensajes y estén en el perímetro). Como esta tecnología está en fase de implantación hay puntos en los que la presencia de receptores ADS-B es muy baja o nula, por tanto en esos puntos el seguimiento de los vuelos se hace más complejo o incluso llegando a ser imposible.

Para tener una idea clara de los datos que se obtienen de las antenas ADS-B vamos a facilitar una tabla en la Figura 2.5 con los datos extraídos y su descripción. En esta podremos ver algunos de los atributos que podemos extraer a partir de los receptores ADS-B, estos necesitaran ser tratados posteriormente.

Etiqueta	Tipo	Descripción
Id	String	La siguiente columna corresponde al identificador del receptor ADS-B
Timestamp	Int	Indica la marca temporal en la cual se extrajo el dato.
Latitude	Float	Indica la latitud de la aeronave en el momento de obtención del dato.
Longitude	Float	Indica la longitud de la aeronave en el momento de obtención del dato.
Altitude	Float	Indica la altitud de la aeronave en el momento de obtención del dato.
Speed	Int	Indica la velocidad a la que iba la aeronave en el momento de obtención del dato.
vspeed	Int	Indica la velocidad vertical de la aeronave en el momento de obtención del dato.
quawk	Int	Código marcado por el piloto en el transpondedor por cada vuelo.
track	Int	Angulo en la que se encuentra la aeronave.
ground	Boolean	Si la aeronave esta tocando suelo o no.
leg	String	Identificador del vuelo.

Figura 2.5: Ejemplos de atributos extraídos por un dispositivo ADS-B.

2.3. Estimación tiempos de llegada

En esta sección vamos a realizar un análisis de alguna de las soluciones que se han ido dando a esta problemática. Al tratarse de una tecnología novedosa que esta aún en proceso de maduración, la mayor parte de las propuestas surgen en el ámbito de la investigación, y en muchos casos con financiación privada como pueden ser compañías como Boeing o Airbus.

Application of Machine Learning Algorithms to Predict Flight Arrival Delays. La primera propuesta se expone en un informe técnico de la Universidad de Stanford por Nathalie Kuhn y Navaneeth Jamadagni [33] en el que intentan dar una solución al problema mediante el uso de varias técnicas de aprendizaje automático.

El *dataset* que utilizan está poblado con información del tráfico aéreo de Estados Unidos del año 2015 y cuenta con 5 millones de muestras divididas en 30 atributos, entre los que se encuentran atributos relativos a:

- Información sobre el vuelo.
- Información sobre el origen y el destino.
- Información sobre la salida.
- Información sobre el transcurso del vuelo
- Información sobre la llegada.
- Información detallada sobre retrasos o cancelaciones.

El fin de este proyecto era predecir si un vuelo iba a sufrir retrasos, para lo que de los 30 atributos que tenían disponibles solo usaron 13: *Month, Day, Day of the week, Flight Number, Origin airport, Destination Airport, Scheduled departure, departure delay, taxi-out, distance, Scheduled Arrival.*

Para el entrenamiento de los modelos seleccionaron 100.000 registros de los 5 millones y seleccionaron 50.000 de vuelos con retraso y otros 50.000 de vuelos que llegaron a su hora, todo esto de manera aleatoria.

En este estudio usaron 3 modelos para realizar las predicciones:

1. Árbol de decisión: La idea detrás de este algoritmo es construir un modelo de árbol desde la raíz hasta los nodos de las hojas. Los nodos reciben una lista de entradas y el nodo raíz recibe los ejemplos provenientes del *dataset* de entrenamiento. Cada nodo verifica el valor de uno de los parámetros, esto divide los datos de entrada entre sus diferentes ramas en función del mismo. Estos subconjuntos se convierten entonces en entradas de los nodos hijo, donde hacen la pregunta sobre las otra característica. Esto se hace para reducir la incertidumbre asociada a la predicción de cada atributo. La dificultad de este algoritmo reside en saber en que orden debe verificar el valor de los distintos atributos o características.

2. Regresión Logística: Es un algoritmo simple de clasificación que usa hipótesis. Consiste en predecir el resultado de una variable categórica (variable que puede adoptar un número limitado de categorías) en función de las variables independientes.

$$h\theta(x) = g\theta^T x = \frac{1}{1 + e^{-\theta^T x}} \tag{2.1}$$

3. Redes Neuronales: Se construye apilando múltiples neuronas en capas para producir un resultado final. Disponen de una capa inicial llamada capa de entrada y una final llamada capa de salida, entre ambas hay diferentes cantidades de capas llamadas capas ocultas. Las neuronas disponen de una función de activación (entre las más importantes están, Sigmoid, Relu, tanh). Algunas redes disponen de diferentes pesos y sesgos por cada capa. El objetivo de una red neuronal es aprender de los parámetros de tal manera que el resultado esperado sea el mismo que el resultado verdadero.

Los resultados obtenidos por este estudio han sido los siguientes (Figura 2.6):

	Predicted Class 0 (on-time)			**Predicted Class 1 (delayed)**		
	Decision Tree	Logistic Regression	Neural Network	Decision Tree	Logistic Regression	Neural Network
True Class 0 (on-time)	13,986	13,907	13,733	1,013	1,092	1,266
True Class 1 (delayed)	1,740	1,670	1,531	13,261	13,331	13,470
precision	0.89	0.89	0.90	0.93	0.92	0.91
recall	0.93	0.93	0.92	0.88	0.89	0.90
f1-score	0.91	0.91	0.91	0.91	0.91	0.91

Figura 2.6: Resultados obtenidos en el artículo 1 [33].

Como podemos ver, el que ha dado unos valores más ajustados ha sido el modelo de árbol de decisión, aunque los demás no se han alejado mucho de la estimación.

Predicting Estimated Time of Arrival for Commercial Flights.

El siguiente documento que vamos a analizar escrito por Samet Ayhan, Hanan Samet ambos de la Universidad de Maryland y Pablo Costas del equipo de investigación europeo de la compañía Boeing [36]. Los datos usados para este experimento han sido recopilados por el proveedor de datos ANSP, ENAIRE. Han contado con un total de 80.784.192 puntos de trayectoria extraídos de vuelos realizados en España en noviembre del 2016. Los atributos que contienen los datos son los siguientes: Número del vuelo, Aeropuerto origen, Aeropuerto destino, Fecha, Hora, Velocidad en los vectores X, Y, Z, Latitud, Longitud, Altitud. Aparte de estos datos cuentan con datos meteorológicos proporcionados por la *National Oceanic and Atmospheric Administration* (NOAA), los datos extraídos cuentan con 40 atributos entre los que se incluyen datos atmosféricos, nubes, etcétera. Un total de 80 Terabytes de datos. Los datos de los aeropuertos son cedidos por EUROCONTROL bajo el programa SESAR anteriormente mencionado; un total de 1.252.571 registros entre los que podemos encontrar atributos como Fecha, Número de vuelo,Aeropuerto origen, Aeropuerto destino, Tipo de nave, Hora de salida real, Hora de llegada real, Hora de salida programada y Hora de llegada programada.

Los algoritmos usados en el estudio fueron los siguientes:

- Lineales:
  1. Regresión Lineal
  2. Regresión de Lasso
  3. Regresión de red elástica.
- No Lineales:
  1. Árboles de decisión.
  2. Support Vector Regression (SVR)
  3. k-Nearest Neighbors (KNN)
- Redes Neuronales Recurrentes:
  1. Long Short-Term Memory (LSTM): Tipo especial de redes recurrentes, se caracterizan porque la información puede persistir. Por tanto pueden "recordar" estados previos y usar la información para decidir cuál será el siguiente estado. Muy usadas en series cronológicas, estas pueden aprender largas secuencias de información por tanto podemos decir que tienen "memoria" de largo plazo.
- Otros:
  1. Adaptive Boosting (AdaBoost): Consiste en un meta estimador, el cual está basado en la idea de tener un grupo de expertos que emite una decisión por mayoría. Al grupo de "expertos" se les conoce como *ensemble* y representan al clasificador fuerte. A los expertos que conforman el *ensemble* se les denomina clasificadores débiles, que individualmente presentan una precisión menor. Su funcionamiento consiste en ponderar de manera diferente los ejemplos durante la fase de entrenamiento, aquellos clasificados de manera errónea reciben una ponderación más alta en la siguiente iteración, por tanto los que fueron correctos recibirán una ponderación menor. Esto hace que el algoritmo se centre en los casos que todavía no consigue clasificar correctamente.
  2. Gradient Boosting Machine (GBM): Es una técnica utilizada en el aprendizaje automático mediante regresión y clasificación estadística. Produce un modelo predictivo que forman un conjunto de modelos denominadas débiles, como en el caso anterior, estos suelen ser arboles de decisión. Funcionan de manera similar al algoritmo visto anteriormente ya que son de la familia *Boosting*, solo cambia la función.
  3. Random Forest Regression (RF): Es una técnica que combina varios arboles de decisión, de manera que cada árbol depende de los valores de un vector aleatorio que es probado de manera independiente y con la misma distribución en cada uno de los arboles. Una vez tiene toda la colección de arboles, usa el promedio de sus valores para así obtener el valor predicho.

4. Extra Trees Regression (ET): También conocidos como *Extremly Randomized Trees* son similares a los Random Forest pero a la hora de la construcción del ensamble varia la manera de construirlos para añadir un factor mas alto de aleatoriedad [17].

Model	LEBL LECO	LEBL LEMG	LEBL LEVX	LEBL LEZL	LEMD LEAM	LEMD LECO	LEMD LEJR	LEMD LEMH	LEMD LEPA	LEMD LEVX
HA	5.590140	5.498889	5.555261	4.911651	3.265500	4.000898	3.311410	3.953404	3.787092	4.666325
LR	6.565834	5.583422	6.669648	5.240185	4.848514	4.612558	4.167661	4.433904	4.943242	5.183345
LASSO	5.423176	5.328147	4.566031	4.620595	2.979159	3.939358	3.061945	3.618408	3.830357	4.676572
EN	5.129900	5.328147	4.510600	4.598869	2.979159	3.972221	3.078638	3.618059	3.759207	4.562907
KNN	4.315834	4.041417	4.080471	3.768015	3.768015	3.472992	2.837109	3.109983	3.633360	3.409278
CART	6.208306	5.228796	5.098875	4.717167	4.527573	4.439111	3.729169	3.790538	4.691645	4.175965
SVR	4.850719	5.036133	4.276852	4.320527	2.914718	3.575904	3.001464	3.199346	3.536542	4.151692
AB	**3.991038**	3.790171	4.244822	**3.620588**	**2.840969**	**3.128822**	2.602778	2.987388	3.347820	**3.092941**
GBM	3.993338	**3.430164**	4.068700	3.630516	3.045774	3.276929	**2.589360**	**2.908439**	**3.344821**	3.174045
RF	4.308547	3.841439	4.103479	3.706125	3.016808	3.345249	2.740151	2.910654	3.5721842	3.437595
ET	4.171390	3.538316	4.266809	3.731383	3.081153	3.396700	2.630542	3.028200	3.6377407	3.436975
LSTM	7.586650	5.674340	**3.708545**	4.673223	2.932544	3.714577	2.604398	3.988213	3.789673	4.311236

Figura 2.7: Resultados algoritmos predicción artículo 2 [36].

Los resultados de la tabla se corresponden al RMSE (Raíz error cuadrático medio) de todos los algoritmos. Sirve de método de evaluación para saber cuanto se aleja la predicción del valor real.

El resultado del estudio es muy bueno, ya que se consigue un modelo que es capaz de predecir el tiempo de llegada de cualquier vuelo comercial en España con un margen de error de 4 minutos, esto hace que supere el sistema de predicción de EUROCONTROL.

Development of a predictive model for on-time arrival flight of airliner by discovering correlation between flight and weather data.

El tercer documento que vamos a analizar es de Noriko Etani de la Universidad de Kyoto [16] el cual pretende demostrar la relación directa entre los fenómenos atmosféricos y los tiempos de llegada. Los datos que se usan en este documento provienen de FLIGHTSTATS y los datos meteorológicos provienen de la Agencia Meteorológica Japonesa. No disponemos de información sobre la cantidad de datos recogidos, pero sí de los atributos usados para el modelo, que son los siguientes: Fecha de salida, Fecha de llegada, latitud del aeropuerto de origen, latitud del aeropuerto de destino, longitud del aeropuerto de origen, longitud del aeropuerto de destino, temperatura, humedad, dirección del viento, presión a nivel del mar, resultados del vuelo.

Los algoritmos usados son los siguientes:

1. Support Vector Machine (SVM).
2. Gradient Boosting.
3. Random Forest.
4. AdaBoosting.
5. Decision Tree.

Tras el estudio de los datos en diferentes escenarios como podemos ver en la tabla de la Figura 2.8 el modelo que mejores predicciones nos ofrece es Random Forest.

Evaluation	Accuracy	Precision	Recall	F-score
Grandient Boosting with weather data	0.70	0.74	0.89	0.81
Random Forest with weather data	0.72	0.74	0.95	0.83
Grandient Boosting without weather data	0.64	0.77	0.72	0.74

Figura 2.8: Resultados algoritmos predicción del artículo 3 [16].
Roberto Calvo-Palomino, Fabop Roccoatp, Blaz Repas, Domenico Giustino,Vincent Lenders

Nanosecond-precision Time-of-Arrival Estimation for Aircraft Signals with low-cost SDR Receivers.

El siguiente estudio escrito por Roberto Calvo, Fabio Ricciato, Blaz Repas, Domenico Giustiniano, Vincent Lenders [9] se basa en intentar predecir el tiempo de llegada con una precisión de nanosegundos. Los datos que analizan provienen de diferentes fuentes, todas ellas de antenas ADS-B. Los atributos presentes en los datos son los habituales en ADS-B.

Aunque este planteamiento este orientado a la predicción mediante la frecuencia y otros datos que recibe el avión por tanto los algoritmos pueden variar de nuestro enfoque, no obstante es interesante analizarlos.

- CorrPulse
- PeakPulse
- Correlation with whole-packet
- Algoritmo basado en Dump 1090

No son algoritmos corrientes ya que han sido desarrollados especialmente para este problema. El algoritmo que mejor resultado ha dado es el llamado CorrPulse/S con un porcentaje de acierto del 83 %

Target tracking and Estimated Time of Arrival (ETA) Prediction for Arrival Aircraft.

En el siguiente documento escrito por Kaushik Roy, Benjamin Levy, Clarie J. Tomlin [28] encontraremos un problema similar al que vamos a plantearnos nosotros, los datos analizados por ellos han sido obtenidos gracias a *Sensis Corporation at Saint Louis Lambert International Airport*. No nos proporcionan información sobre el tipo de datos que analizan, pero suponemos que serán parecidos a los obtenidos por un sistema ADS-B.

Los algoritmos usados para el estudio de este problema han sido los siguientes:

- IMM (*Interacting Multiple Model*) (Markov)
- IMM (Autonomous)
- Particle Filter (Markov) N=10/20/100
- Particle Filter (Autonomous) N=10/20/100

Como podemos observar, no son algoritmos comunes utilizados en este tipo de problemas, son adaptados a la problemática y a los datos que disponen. La mejor solución es el algoritmo IMM con transiciones automatizadas, siendo su resultado el poder predecir con una antelación de 15 minutos y con un error de 30 segundos.

Capítulo 3

Gestión del proyecto

En este capítulo vamos a detallar en profundidad la metodología seguida a lo largo del desarrollo del proyecto, justificando así su utilización. Primero describiremos la metodología usada, después vamos a detallar la planificación temporal realizada y los costes previstos asociados a esta. Debemos tener en cuenta que al tratarse de un proyecto de investigación, la planificación y estimación de los costes será más compleja de llevar a cabo, ya que hay un alto grado de incertidumbre.

3.1. Metodología

El desarrollo de software de manera ágil no es algo nuevo, pero sí ha ido aumentando su uso a lo largo del tiempo. Se trata de usar métodos basados en el desarrollo iterativo e incremental, todo esto bajo la filosofía de requisitos y soluciones que evolucionan mediante la colaboración entre equipos multifuncionales y autorganizados. La historia de los marcos de trabajo ágiles se remonta al uso de *Lean Manufacturing* de Toyota [41] (1940). Ese fue el comienzo de lo que años más tarde y después de sufrir varias transformaciones por diferentes vertientes, daría lugar al nacimiento del Manifiesto Ágil [3] en el año 2001. Desde entonces el agilismo ha ganado mucha fuerza y ha seguido evolucionando.

Los marcos de trabajo ágiles surgen ante la necesidad de dar solución a un entorno con muchas variables y un nivel alto de incertidumbre, intentando reducir la franja existente entre lo que el cliente desea y lo que le ofrecemos. Según el Informe Caos del año 2015 [1] el cual compara el desempeño de numerosos proyectos, de los cuales realiza una comparativa entre proyectos ágiles y proyectos en cascada que pertenecen a la metodología tradicional, podemos ver los resultados obtenidos en (Figura 3.1):

[1] https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf

CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

Figura 3.1: Chaos Report 2015 [10]

Como podemos observar respecto a todos los proyectos analizados, los proyectos exitosos en metodología de cascada son de solo un 11 %, mientras que en metodologías ágiles llegan hasta el 39 % esto es debido que, los proyectos ágiles se centran en ofrecer lo que el cliente quiere, minimizando todo lo posible las desviaciones de la visión original. Otro dato interesante es el número de proyectos fallidos mientras que en Ágiles tenemos un 9 % en metodologías tradicionales esto asciende hasta el 29 %, es un número muy elevado de proyectos fallidos.

Analizando el gráfico podemos darnos cuenta que el tamaño del proyecto también influye a la hora de realizar estos de manera exitosa. En proyectos pequeños las metodologías ágiles y la de cascada tienen valores mas cercanos en cuanto a su tasa de éxito, la cual es de un 58 % en ágiles frente al 44 % en las de cascada. En cuanto a la columna en amarillo denominada *Challenged* se refiere a proyectos que no esta muy claro si tuvieron éxito o fracasaron. En ese ámbito sigue siendo menor el porcentaje asociado a los proyectos ágiles.

Con este informe podemos sacar en claro que cuanto más grande es el proyecto mejor será el desempeño de una metodología ágil frente a una tradicional. Como comentamos antes es debido a que cuanto mayor es el proyecto mayor es el nivel de incertidumbre de

este.

Las marcos de trabajo ágiles se fundamentan en cuatro pilares básicos recogidos en el manifiesto ágil:

1. Individuos e interacciones sobre los procesos y herramientas: Un proyecto que sigue una metodología ágil debe tener como meta alcanzar el máximo nivel de calidad posible, basándose en el desempeño y conocimientos del equipo y no tanto a la disponibilidad de las herramientas para llevarlo a cabo.

2. Software funcionando sobre documentación extensiva: Esto nos permite tanto a nosotros como al cliente hacerse una idea de cómo será el producto final y precisamente ahí es donde estrechamos el gap gracias a la retroalimentación del cliente. Este feedback es lo esencial en proyectos desarrollados con esta metodología, ya que nos permite hacernos una idea más precisa de las necesidades del cliente, y en muchos casos ayuda al cliente a saber mejor qué es lo que quiere.

3. Colaboración con el cliente sobre negociación contractual: En este tipo de metodologías es muy importante el compromiso, estos proyectos están en continua evolución y es muy difícil definir unos requisitos iniciales que definan cómo sera el producto al finalizar el proyecto.

4. Respuesta ante el cambio sobre seguir un plan: En proyectos cuyo grado de incertidumbre es muy alto con requisitos cambiantes y que están predestinados a que evolucionen constantemente y de manera muy rápida, es necesario tener capacidad de responder ante estos cambios.

Existen numerosas propuestas de marcos de trabajo entre los más populares se encuentran Scrum[2], Kanban[3] o XP [4] (eXtreme Programming). Para este proyecto, al tratarse de un proyecto de investigación el cual tiene un alto grado de incertidumbre y requisitos cambiantes, nos hemos decantado por el marco de trabajo Scrum por considerarlo idóneo para la organización del proyecto dada su naturaleza, y porque el equipo de trabajo está plenamente familiarizado con él al estar certificado como Scrum Master por la organización European Scrum. Además es uno de los *frameworks* más usados actualmente en la industria del desarrollo. Todo lo que vamos a exponer a continuación es extraído de la Guía de Scrum [26] creada por Ken Schwaber y Jeff Sutherland.

3.1.1. Scrum

Originariamente Scrum se caracteriza por los siguientes reglas:

- Adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto. Esto nos permitirá hacer micro entregas del producto,

[2]https://www.scrum.org/resources/blog/que-es-scrum
[3]https://kanbantool.com/es/metodologia-kanban
[4]http://www.extremeprogramming.org/

lo cual nos facilitará estar más cerca de la necesidades del cliente sin desviarnos del objetivo final.

- Basar la calidad del resultado más en el conocimiento tácito de las personas y en equipos auto organizados, que en la calidad de los procesos empleados.
- Solapar las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o en cascada. Así con cada entregable estaremos añadiendo valor desde la primera entrega.

Los proyectos desarrollados bajo este *framework* comienzan con una percepción general del proyecto y a partir de ella se detalla, cada vez más, empezando por un nivel de abstracción alto a uno mas bajo. Esta metodología se caracteriza por el desarrollo cíclico. Cada ciclo en Scrum es denominado, Sprint que forma parte de los eventos en Scrum, y finaliza con una entrega del producto, que tiene que aportar valor al cliente. En estos ciclos no se debe exceder la duración de 4 semanas por cada Sprint, así evitaremos distanciarnos de la visión del cliente y podremos aportar valor lo antes posible.

En Scrum tenemos varios roles (Figura 3.2) y cada uno tiene sus particularidades:

ROL	Numero de personas	Desempeño
Product Owner	1	• Tiene la Autoridad sobre **QUE** hay que hacer. • Conocimiento total sobre el producto y las necesidades del cliente. • Capaz de crear una visión del producto. • Colabora con Stakeholders y el equipo. • Crea, mantiene y prioriza el Product Backlog (Pila de trabajo pendiente) • Participa en reuniones. • Ayuda a entender y hacer más transparente el entendimiento sobre el Product Backlog.
Scrum Master	1	• Actúa como facilitador • Siempre al servicio del equipo. • Elimina impedimentos que surjan. • Mantiene contacto con el Prod.Owner y el equipo. • Lidera y guía al equipo (sin imponerse) • Gestiona y se asegura de que se lleven a cabo todas las ceremonias.
Equipo de desarrollo	3-9	• Trabajan de manera unitaria y colaborativa. • Son autoorganizados por tanto tienen autoridad para decidir como cumplir los objetivos. • Se comprometen a la entrega de un incremento del producto. • Administran el Sprint Backlog • Participan en las reuniones diarias.

Figura 3.2: Roles dentro de Scrum.

En Scrum existen una serie de eventos:

- **Sprint**: Tiene asignado un *time box* menor que 4 semanas (1 mes) al final del sprint se genera un incremento funcional del producto, que puede ser o no entregado al cliente. Los Sprint siguientes serán siempre de la misma duración.
- **Sprint Planning**: Reunión en la que se decide el alcance del próximo Sprint, en ella participa todo el Scrum Team, se debe realizar el comienzo de cada Sprint nuevo y puede durar hasta 8h para un Sprint de un mes. Definimos el qué haremos y el cómo lo haremos.
- **Daily Scrum**: Reunión diaria, o conocida también como reunión de sincronización. En esta reunión asisten el equipo de desarrollo y, opcionalmente el Scrum Master. El propósito principal es alinear y planear el trabajo de las próximas 24h. La duración máxima de este evento es de 15 minutos, a la misma hora. Cada miembro del equipo de desarrollo contesta a las siguientes tres preguntas:
  - ¿Qué hice ayer por el equipo?
  - ¿Qué haré hoy por el equipo?
  - ¿Qué bloqueos me impiden conseguirlo?

  Gracias a esta serie de pasos permitirá tener los bloqueos localizados y solucionados con la mayor brevedad, lo cual nos permitirá avanzar a un ritmo constante. En nuestro caso y teniendo en cuenta el tipo de proyecto que es, las reuniones daily se realizaran una vez a la semana.
- **Sprint Review**: En este evento asisten tanto el Scrum Team como el Product Owner y los Stakeholders. Se muestran las tareas finalizadas a los Stakeholders y se responden todas las dudas que se planteen. Este evento se realiza al finalizar un Sprint, dura como máximo 4h para un Sprint de 1 mes.
- **Sprint Retrospective**: Reunión que tienen el Scrum Team, en esta se tiene la posibilidad de analizar el desempeño del último sprint y crear un plan de mejoría. Se realiza después de cada Sprint Review y antes de los Sprint Planning. Duración máxima de 3h para un Sprint de 1 mes. Las cuestiones que se abordan son:
  - ¿Qué salió bien?
  - ¿Qué debemos mejorar como equipo?
  - ¿Qué debemos mejorar como organización?

  Cabe destacar que en estas reuniones se busca la mejora como conjunto y el avance del equipo, no se trata de acusar a alguien de su rendimiento o realzar los puntos negativos.

Y por último hablaremos de los artefactos:

- **Product Backlog**: Es una lista ordenada que contiene todo lo necesario para el producto, por lo que es la única fuente de requisitos. Su creador es el Product Owner y es el que la actualiza. Se prioriza siempre los trabajos que aporten mas valor, de acuerdo a la visión de negocio y las necesidades del cliente.
- **Sprint Backlog**: Elementos seleccionados del Product Backlog para el alcance de este Sprint. Una manera fácil de tener una imagen del avance del equipo. Gestionada por el equipo de desarrollo.
- **Incremento**: Suma de los elementos del Product Backlog durante el Sprint y todos los anteriores ya completados. Al final de cada Sprint se debe obtener un nuevo incremento.

3.2. Planificación

La planificación de este proyecto se ha realizado bajo el marco de trabajo Scrum. Este proyecto se ha planificado para tener una duración de 6 sprints que, a su vez estos tendrán una duración de 4 semanas por cada sprint. Para la gestión de las tareas del proyecto se ha utilizado la herramienta Trello, que simula un tablero Kanban. En él podremos encontrar los anteriormente citados, Product Backlog, Sprint Backlog. Una vez tenemos las historias de usuario divididas y creadas, se les asigna una fecha para acabar y se definen los criterios de aceptación de cada historia, para que se considere, esta tarea como hecha. Le otorgamos unos puntos llamados puntos de historia que usaran la escala de Fibonacci. En nuestro caso ira desde el 3 hasta el 13 considerando tareas más grandes como épicas que necesitan ser fragmentadas en tareas más pequeñas. Cabe destacar que estos puntos son meramente informativos para que los miembros del grupo estimen como de grande es una tarea, en ningún caso esos números corresponden a las horas necesarias para la realización de la tarea en cuestión.

Como hemos comentado anteriormente, este proyecto tiene la particularidad de ser un proyecto de investigación, por tanto, aunque se usen metodologías ágiles para su desarrollo, sigue siendo complejo de planificar y estimar, ya que la mayoría de las tareas serán dedicadas al estudio y análisis de la problemática que hay alrededor de la gestión aérea y cómo poder mejorar esta. A lo largo del desarrollo de este proyecto se ha contemplado un parón dentro de la planificación correspondiente al periodo de exámenes de la Universidad, ya que este trabajo se ha realizado simultáneamente con el último curso del grado. Para aclarar la disposición de los sprints y los periodos temporales a los que corresponde cada uno, vamos a ver la (Figura 3.3):

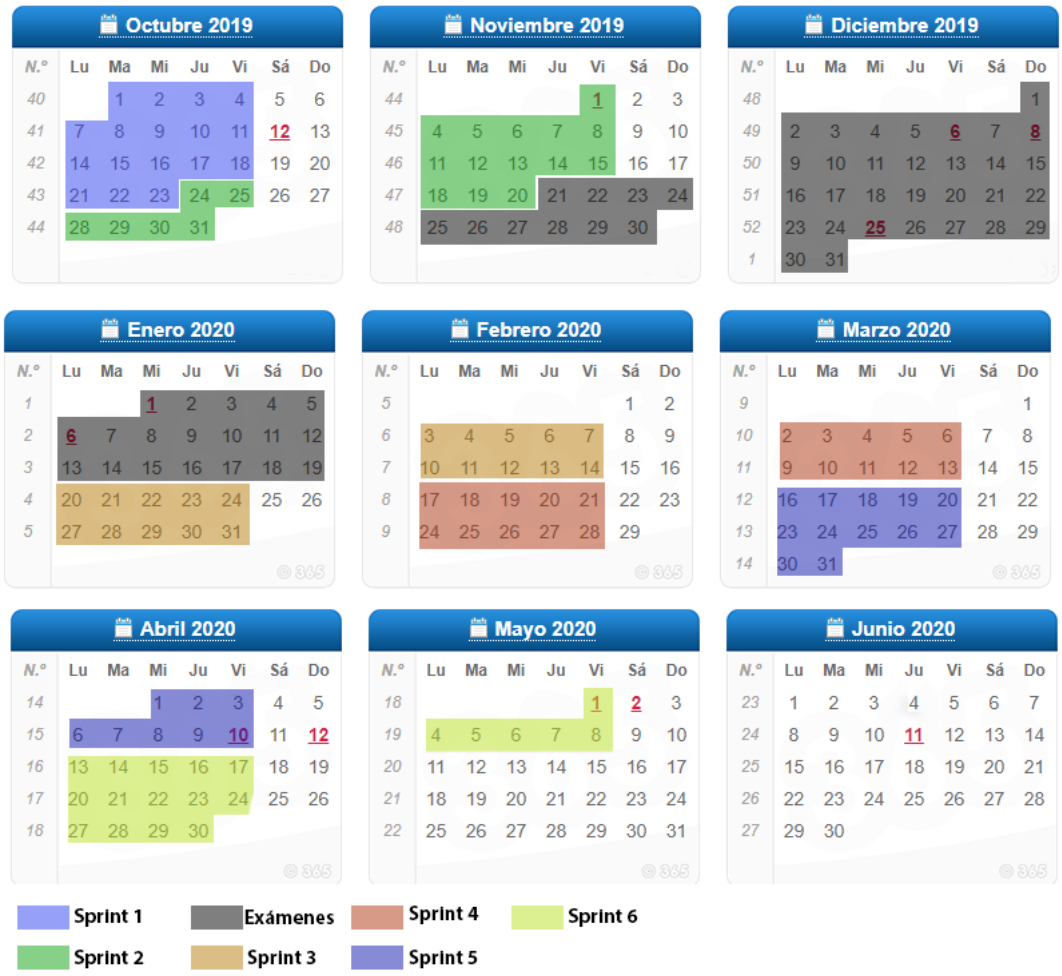


Figura 3.3: Calendario Sprints.

Una vez fijadas las fechas de cada Sprint, y obtenidas las historias de usuario con su correspondiente ponderación en forma de puntos de historia, solo hacía falta estimar las horas que nos llevaría realizar cada tarea. En Scrum un método habitual de estimación es basarse en experiencia de antiguos proyectos. En nuestro caso no contamos con experiencia previa en proyectos similares, por tanto es normal que la estimación pueda desviarse algo mas de lo normal. Esto lo podremos controlar en cierto modo gracias a que en esta metodología lo habitual es saber cuantos recursos tenemos y cuando queremos acabar el proyecto para así poder estimar hasta donde puede abarcar el alcance. Y no como las tradicionales que cuentas con el tiempo y el alcance y lo que debes estimar es el coste. Con esto podremos ir adaptando el alcance de nuestro proyecto y esto permitirá alcanzar nuestro objetivo de manera mas certera. Por tanto la disposición horaria de cada tarea quedaría de la siguiente manera (ver Figura 3.4) a (Figura 3.9).

Como el proyecto se ha realizado conjuntamente con las clases, los primeros sprint tienen menos carga en cuanto a horas. Me gustaría volver a recalcar, que es muy importante que no hagamos una asociación entre horas y puntos de función ya que no la tienen, estos sirven solo como mero indicador de la magnitud de la tarea a nivel organizativo para el equipo. Para el primer sprint después de obtener las historias de usuario obtuvimos esta disposición, debemos tener en cuenta que el apartado de *Coste personal Sprint* corresponde a lo que costaría ese sprint con las horas estimadas en función del sueldo:

SPRINT 1

Tarea	Puntos de funcion previstos	Fecha ini	Fecha Fin	Horas	Coste personal Sprint
Introduccion a BDD seriestemporales	5	01/10/2019	07/10/2019	8	480
Identificacion de BBDD seriestemporales	5	07/10/2019	09/10/2019	8	480
Analisis de BBDD Seriestemporales	5	09/10/2019	16/10/2019	8	480
Conclusion de Analisis deBBDD Series te	3	16/10/2019	18/10/2019	6	360
Documentar BBDDTemporales	5	18/10/2019	23/10/2019	9	540
Pts.Total	23			Total:	2340 €
Horas totales	39				

Figura 3.4: Planificación temporal y estimación de costes del Sprint 1.

El primer sprint tendrá 4 historias de usuario y una quinta que corresponde a la documentación de las anteriores tareas. El objetivo del sprint es el estudio de las bases de datos de series temporales y cuáles se adecuan más a la necesidad que tenemos de operar con series temporales. El incremento que obtenemos es el estudio de las bases de datos plasmado en el documento.

SPRINT 2

Tarea	Puntos de funcion previstos	Fecha ini	Fecha Fin	Horas	Coste personal Sprint
Introduccion problemáticagestion de traf	5	23/10/2019	28/10/2019	9	618,75
Efectos de los retrasos aereos enEuropa	3	28/10/2019	30/10/2019	6	412,5
Estudio de receptores ADS-B	5	30/10/2019	06/11/2019	10	687,5
Estado del arte sobre prediccionde tiemp	5	06/11/2019	14/11/2019	10	687,5
Documentar ATM	3	14/11/2019	20/11/2019	5	343,75
Pts.Total	21			Total:	2750 €
Horas totales	40				

Figura 3.5: Planificación temporal y estimación de costes del Sprint 2.

En este segundo Sprint podemos ver una disposición similar al anterior de 4 historias más una de documentación. El objetivo de este es el estudio del entorno en el que se encuentra el problema de la gestión trafico aéreo y de cómo esta se realiza. Una vez finalizado este Sprint obtendremos un estudio de la gestión aérea plasmada en el documento.

Sprint 3

Tarea	Puntos de funcion previstos	Fecha ini	Fecha Fin	Horas	Coste personal Sprint
Adecuación de los datos al problema.	13	20/01/2020	06/01/2020	40	2750
Estudio, modelización y evaluación de al	8	06/02/2020	14/02/2020	18	1237,5
Pts.Total	21			Total:	3987,5 €
Horas totales	58				

Figura 3.6: Planificación temporal y estimación de costes del Sprint 3.

Este tercer Sprint correspondiente con la época posterior a exámenes, por lo que en adelante se podrá dedicar más tiempo al desarrollo del proyecto. En este Sprint se van a abordar historias muy grandes, de ahí que sean solo dos ya que se estima que nos llevarán más tiempo. Las historias se corresponden a la adecuación, pre-procesado de los datos y el estudio del modelo para determinar los datos que necesitará para llevarse acabo. Una vez finalizado obtendremos datos válidos para nuestro problema y un estudio que avale la transformación de los datos necesarios para el modelo.

Sprint 4

Tarea	Puntos de funcion previstos	Fecha ini	Fecha Fin	Horas	Coste personal Sprint
Sistema de Predicción.	13	14/02/2020	28/02/2020	30	2062,5
Conclusiones del estudio y modelización	13	28/02/2020	13/03/2020	30	2062,5
Pts.Total	26			Total:	4125 €
Horas totales	60				

Figura 3.7: Planificación temporal y estimación de costes del Sprint 4.

El sprint 4 probablemente sea el que más esfuerzo suponga, ya que en este se desarrollará el sistema de predicciones y se pondrá cierre al estudio y modelización del problema. Una vez acabado este sprint obtendremos el modelo predictivo funcional.

Sprint 5

Tarea	Puntos de funcion previstos	Fecha ini	Fecha Fin	Horas	Coste personal Sprint
Estudio componentes de dash	5	13/03/2020	27/03/2020	9	618,75
Creacion de componentes interactivos	8	27/03/2020	03/04/2020	10	687,5
Integracion modelo ML	3	03/04/2020	08/04/2020	7	481,25
Creacion Dash	5	08/04/2020	10/04/2020	9	618,75
Documentar Dashboard	3	10/04/2020	15/04/2020	9	618,75
Pts.Total	24			Total:	3025 €
Horas totales	44				

Figura 3.8: Planificación temporal y estimación de costes del Sprint 5.

El Sprint 5 se corresponde a la construcción de un cuadro de mandos, capaz de reflejar mediante gráficos interactivos el estudio y hacer más fácil la comprensión de los datos. En él tendremos 4 historias y una de documentación.

Sprint 6

Tarea	Puntos de funcion previstos	Fecha ini	Fecha Fin	Horas	Coste personal Sprint
Completar datos del finalizacion de proy	8	15/04/2020	24/04/2020	20	1375
Revision memoria	5	24/04/2020	01/05/2020	15	1031,25
2 da Revision	5	01/05/2020	05/05/2020	12	825
Creacion de presentacion	3	05/05/2020	08/05/2020	12	825
Pts.Total	21			Total:	4056,25 €
Horas totales	59				

Figura 3.9: Sprint 6.

Este sexto y último Sprint se dedicará a completar la documentación y el balance, y realizar las correcciones pertinentes antes de la entrega de la memoria, así como, la creación de una presentación que apoye la documentación.

La suma de horas estimadas de este proyecto se corresponde con 300 horas que son las horas que debería llevar el desarrollo de un Trabajo Fin de Grado.

Pts Totales	136
Horas totales est:	300

Figura 3.10: Horas estimadas/Pts.Completados.

3.3. Presupuesto

Como hemos dicho anteriormente, es bastante difícil realizar un presupuesto en un proyecto de investigación, ya que primero no sabemos qué herramientas vamos a utilizar a lo largo de la investigación, y dónde vamos a encontrar un impedimento el cual nos puede llevar más horas de las previstas. Por tanto, si voy a facilitar los datos correspondientes a la estimación inicial del proyecto, estos son meramente informativos para poder hacernos una idea de los que podría suponer llevar a cabo el proyecto. Además al ser un proyecto de investigación el alcance esta abierto a incorporar nuevos requisitos.

A continuación voy a detallar los costes del proyecto, a modo aclarativo diré que todos los costes a excepcion del coste de los recursos humanos se han calculado en función del tiempo de uso (1 mes, 2 meses) y de la frecuencia de uso (50 %,64 %).

- Costes Software: Como podemos observar en la (Figura 3.11) se detalla toda la información respecto a los costes relativos a software. Como mencionamos antes estos tendrán componentes presupuestados en función del porcentaje de uso y del tiempo de uso.

Software

Programas	Coste mes	%Uso	Meses de uso	Coste real	
Google Colab	0	0		0	
PyCharm	16,58	100	6	99,48	
Photoshop	24,19	100	2	48,38	
Overleaf	0	0		0	
Slack	0	0		0	
Trello	0	0		0	
				Total	***147,86***

Figura 3.11: Coste Software.

- Costes Hardware: En este apartado podemos observar el coste asociado al hardware, cabe destacar que se ha aplicado la misma medida de ponderación del precio según la frecuencia de uso, el tiempo de uso y el coste mensual. En el caso del ordenador primero hemos calculado cuánto nos costaría mensualmente, para esto hemos considerado que un ordenador tiene una frecuencia de reemplazo de 5 años. Además de esto, usaremos un 50 % del tiempo para el proyecto por tanto:

$$650(Coste\ PC)/60\ meses(5\ \text{años}) = 10,83\ Coste/mes.$$

$$10{,}83 * 50\,\%\ de\ uso = 5,41\ \text{Coste Mes*Frec.Uso}$$

$$5,41 * 6(meses\ de\ uso) = 32,49$$

Los gastos corrientes, salvo internet, no son dedicados y por tanto no se incluyen en la estimación.

Comunes + Hardware

Servicios	Coste mes	%Uso	Meses de us	Coste real	
Internet	48	50	6	144	
Ordenador	10,83	50	6	32,49	
				Total	*176,49*

Figura 3.12: Costes Hardware.

- Costes de recursos humanos: Para el cálculo de la recursos humanos hemos usado como referencia el sueldo medio de un científico de datos junior ya que suelen ser los que desempeñan las labores descritas a lo largo de este documento. Por ello, tomamos como referencia los 33.000 €/año[5], este es el coste que supone para la empresa, incluyéndose los costes sociales y el salario del trabajador. Con este dato podemos calcular el coste por hora que serán de 68,75 €/h. Para la obtención del número de horas y el coste de cada etapa nos hemos basado en la planificación realizada anteriormente de los diferentes Sprint, de (Figura 3.4) a la (Figura 3.9)

  Con estos datos podemos calcular el coste total que nos suponen los recursos humanos solo tenemos que aplicar esta fórmula:

$$\text{Coste Salario Total} = \text{Numero Horas} * 68,75\ €/\text{h}$$

Realizando los cálculos obtenemos 300h * 68,75 €/h = 20.625 €.

Ahora que disponemos de todos los datos relacionados con los costes del proyecto vamos a realizar la suma de todas para obtener el precio final del proyecto:

$$\text{Total Proyecto} = \text{C.Hardware} + \text{C.Software} + \text{C.Recuros humanos}$$

$$\text{Total Proyecto} => 176,49 + 147,86 + 20.625 = \mathbf{20.949,35\ €}$$

3.4. Balance

Después de seguir la planificación realizada al principio del proyecto, han surgido una serie de complicaciones que han tenido efectos directos en el proyecto. En el Sprint 4 hemos tenido un bloqueo que no nos permitió acabar el Sprint según lo previsto y se tuvo que extender y añadir un nuevo Sprint en la planificación. Antes de realizar esta extensión tuvimos que evaluar que estuviera dentro de las fechas de entrega, y una vez comprobamos que era posible de realizar, se llevó a cabo. Por tanto, el nuevo calendario quedaría así:

[5]https://www.glassdoor.es/Sueldos/junior-data-scientist-sueldo-SRCH_KO0,21.htm

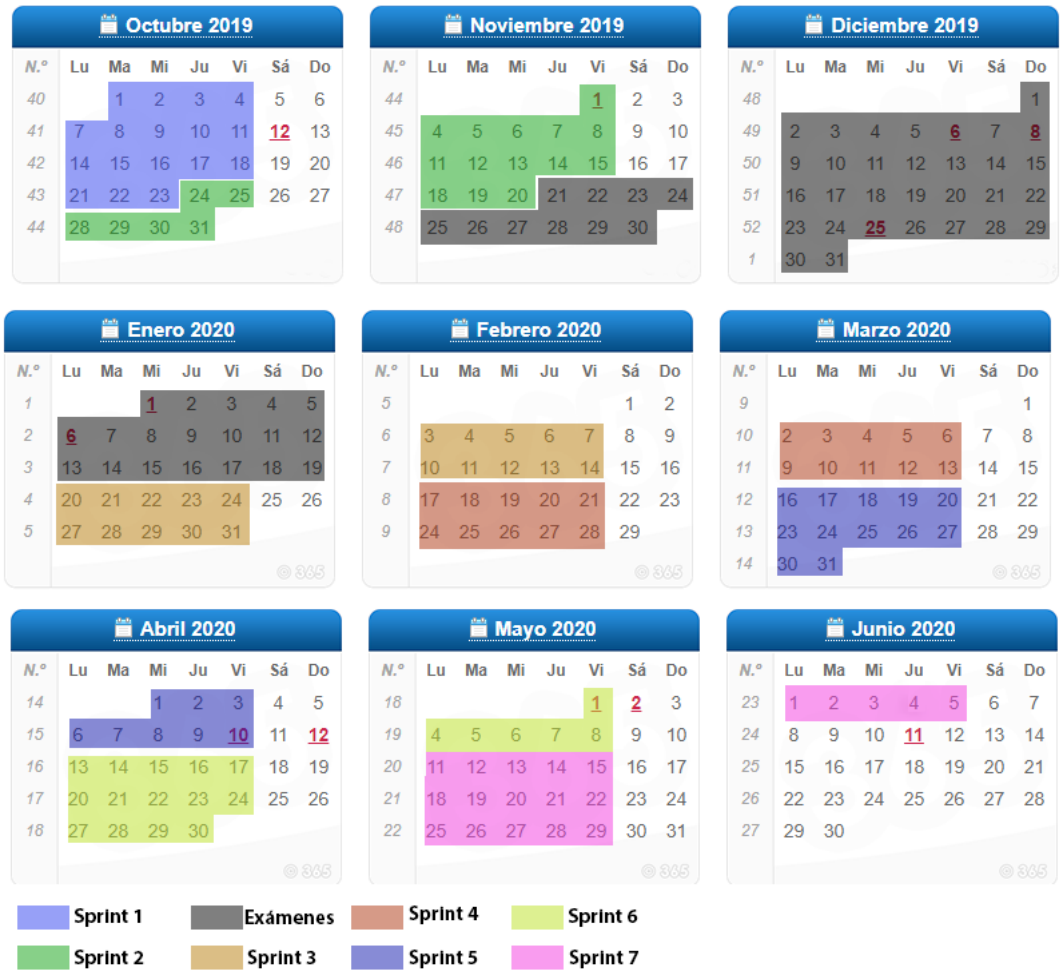


Figura 3.13: Calendario Sprints Modificado.

Obviamente, esto conlleva un incremento de las horas y por tanto de los gastos. El Sprint añadido es el siguiente le damos el nombre de Sprint 5 porque va a continuación del Sprint el cual tuvimos bloqueo, y reubicamos los demás sprint, la labor de este sprint será acabar las tareas inacabadas del anterior y además documentar el proceso.

Sprint 5 EXTRA

Tarea	Puntos de funcion previstos	Fecha ini	Fecha Fin	Horas	Coste personal Sprint
Sistema de Predicción.	13	13/03/2020	27/03/2020	25	1718,75
Conclusiones del estudio y modelización	8	27/03/2020	03/04/2020	15	1031,25
Visualizacion	5	03/04/2020	08/04/2020	15	1031,25
Documentar ML	3	08/04/2020	10/04/2020	3	206,25
Pts.Total	29			Total:	3987,5 €
Horas Reales	70			Coste real Personal Sprint	4812,5 €
Horas totales	58				

Figura 3.14: Sprint 5 Extra.

También se han producido cambios en los gastos tanto de software como de hardware y comunes ya que usaremos los servicios un mes más.

Comunes + Hardware

Servicios	Coste mes	%Uso	Meses uso	Coste real	
Internet	48	50	7	168	
Ordenador	10,83	50	7	37,905	
				Total	*205,905*
				Coste antes	176,49

Software

Programas	Coste mes	%Uso	Meses de u	Coste real	
Google Colab	0	0		0	
PyCharm	16,58	100	7	116,06	
Photoshop	24,19	100	2	48,38	
Overleaf	0	0		0	
Slack	0	0		0	
Trello	0	0		0	
				Total	***164,44***
				Coste antes	147,86

Figura 3.15: Cambio coste SF/HW/Comunes

Esto nos deja con una diferencia de:

$$\text{Desviación Coste HW+SF} = \text{Total Después} - \text{Total Antes}$$

$$\text{Desviación Coste HW+SF} => 370,34 - 324,35 = 45,99€$$

Por tanto, si sumamos el gasto adicional de software, hardware, comunes y le añadimos a esto la suma de las horas extra que ha supuesto el proyecto , teniendo en cuenta que el bloqueo que hemos tenido era muy complejo, ya que es algo nuevo y no hay ninguna referencia a la cual poder acudir, la desviación no ha sido tan grande, en total el coste de esta supone:

$$\text{Desviación Total} => (23.787{,}5 + 370,34) - (20.625 + 324,35) = 3.208{,}49\ €$$

Desviacion coste	3208,495	€
Desviacion horas	46	h

Figura 3.16: Desviación Proyecto.

El coste que nos ha supuesto desarrollar el proyecto en total es de:

$$\text{Coste Total} => (\text{Coste Personal} + \text{Coste HW/otros+SF})$$

$$\text{Coste Total} => (23.787{,}5 + 370,34) = \mathbf{24.157{,}84\ €}$$

Capítulo 4

Series Temporales

En el siguiente capítulo, introduciremos el concepto de serie temporal dentro de nuestro proyecto. Posteriormente realizaremos un estudio de los diferentes algoritmos de aprendizaje automático existentes aplicables a este tipo de datos. Estos han sido recopilados gracias al estudio previo de anteriores soluciones que se han dado al problema. Analizaremos las características de los diferentes modelos para entender su funcionamiento.

4.1. Introducción

Una serie temporal es una secuencia de valores observados en una métrica o sistema en un instante determinado. Cada valor está asociado al instante en que fue observado de forma unívoca, lo que permite ordenar, desde un punto de vista temporal, estos valores dentro de una secuencia continua.

Las series temporales son muy útiles cuando queremos analizar datos en periodos de tiempo. Un ejemplo de uso de serie temporal que todos conocemos son las cotizaciones en bolsa de una determinada empresa. En un gráfico bursátil, sobre el valor de una empresa en bolsa podemos, distinguir dos variables: una es el tiempo y otra es el valor que tiene en el momento de la medición. Además en este ejemplo es siempre a la misma hora, al realizarse el cierre de bolsa diario se hace la toma y se actualizan los resultados. La combinación de tiempo-valor de un conjunto de estas mediciones, como por ejemplo 3 meses, nos muestran cómo ha ido evolucionando el precio de la acción durante ese intervalo temporal y, a partir de ahí se pueden realizar estudios para intentar realizar predicciones sobre el valor de esa empresa en un futuro. Lógicamente estas predicciones tienen sus limitaciones. Para eso sirven las series temporales, para organizar datos en distintos extractos de tiempo y poder analizarlos, estudiar sus tendencias y ver las relaciones que existen entre unos y otros.

4.2. Análisis de modelos de Aprendizaje Automático

En esta sección procederemos a analizar los diferentes modelos existentes para el tratamiento de series temporales. Analizaremos desde el modelo más clásico como es el modelo

ARIMA, hasta uno de los modelos más avanzados hoy en día que es el de las redes neuronales LSTM. Este análisis lo hacemos con el fin de entender el funcionamiento de los distintos modelos, para posteriormente poder aplicar el conocimiento obtenido a la hora de desarrollar nuestro propio modelo de aprendizaje basado en estas técnicas.

4.2.1. ARIMA

El modelo más común en la estadística clásica es el modelo ARIMA (*Auto Regresive Integrated Moving Average*), consolidado por Box y Jenkins en 1976 [6]. Su nombre se debe a los tres componentes que lo forman: el *auto-regresivo*, AR; el *integrado*, I; y el de *medias móviles*, MA. Dentro de la estadística clásica es el modelo más común para predecir valores de series temporales.

El modelo ARIMA permite describir un valor como una función lineal de datos basados en el pasado y errores que se producen por el azar. Además se pueden incluir componentes cíclicos o estacionales. De esta manera el modelo nos permitirá predecir un valor futuro gracias a la información de la que dispone de valores pasados.

Este modelo funciona de la siguiente manera:

1. En la primera fase debemos identificar el modelo ARIMA.
   - Convertir la serie analizada a una serie estacionaria. Las series de estas características muestran una media y varianza constantes a lo largo de una franja temporal.
   - Una vez realizado el paso anterior, tenemos que determinar el modelo, es decir, los ordenes de los parámetros p y q. A su vez estos se corresponden a la estructura autorregresiva y a su media móvil respectivamente.
2. En la segunda fase de selección del modelo provisional de serie estacionaria, obtendremos los parámetros AR y MA, y por tanto su error estándar y los residuos del modelo. Fórmulas de AR y MA:
   AR:
   $$AR(p) \equiv X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + ... + \phi X_{t-p} + a_t \tag{4.1}$$
   ϕ_i: corresponde a los parámetros pertenecientes a la parte autoregresiva del modelo.
   X_t: corresponde a los parámetros pertenecientes a las medias móviles.

   MA:
   $$X_t = a_t - v_1 a_{t-1} \tag{4.2}$$
   v_1: corresponde a la innovación pasada.

   a_t: corresponde a la innovación futura.

   Y_t: corresponde al ruido blanco.

   Determina el valor X en el momento t en función del estado y su primer retardo. Estos modelos se suelen denominar de memoria corta, esto es debido a que como observamos la alteración at que ocurre en el momento t afecta solo a Y_t e Y_{t+1} por tanto su memoria sera de 1 solo periodo [4].

3. En la tercer fase vamos a analizar que los residuos no tengan estructura y que sigan un proceso de ruido blanco, es decir, que contenga aleatoriedad que no haya patrones o aglomeraciones en la representación.

4. En la cuarta fase se obtiene un modelo el cual podrá hacer predicciones sobre los datos entrenados.

   Para facilitar la visualización del funcionamiento del modelo, vamos a ver de manera gráfica las diferentes etapas (ver Figura 4.1):

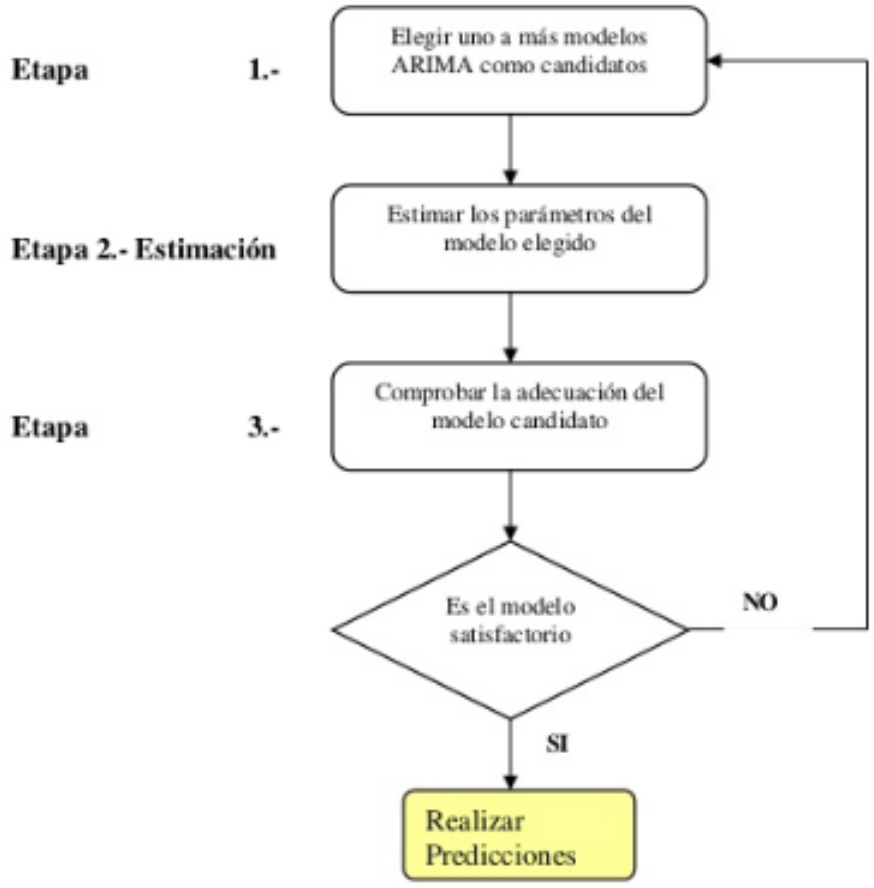


Figura 4.1: Fases ARIMA.

Podemos representar los valores reales representados en azul y la curva que forman representado en morado, y compararlos con la predicción generada por ARIMA fragmento final en rojo. Este método nos permite obtener la distribución de puntos probable en un periodo futuro (ver Figura 4.2):

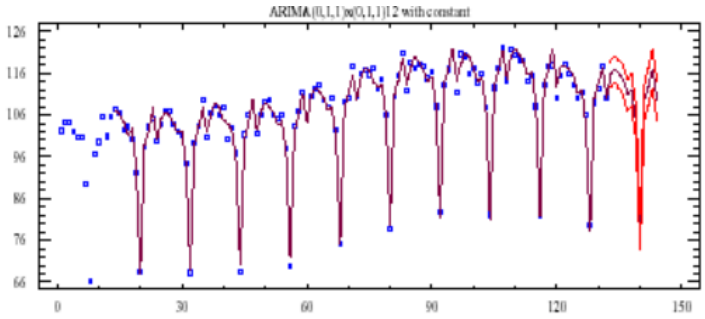


Figura 4.2: Resultado gráfico modelo ARIMA.

4.2.2. Gradient Boosting

El modelo de Gradient Boosting [22] consiste en utilizar un ensemble (conjunto de modelos de aprendizaje automático más simples o "débiles") para abordar tanto problemas de regresión como problemas de clasificación. En general, se entrenan árboles de decisión para usarlos como modelos débiles.

Con el fin de entender el funcionamiento de este modelo, vamos a explicar la la función de perdida, en que consiste el algoritmo de aprendizaje débil, y como aplica el modelo aditivo para evitar la influencia de otros modelos creados.

Función de pérdida La función de pérdida depende directamente del problema que vayamos a tratar, y debe ser diferenciable. Existen diferentes tipos de funciones de pérdida [32] estándar. Entre ellas podemos encontrar el error cuadrático, perfecto para problemas de regresión, perdida logarítmica o para problemas de clasificación.

Algoritmo de aprendizaje débil Como hemos mencionado anteriormente este modelo, usa el árbol de decisión. En concreto se usa el árbol de regresión, ya que estos producen valores para las divisiones cuya salida se puede sumar. Esto se hace con el fin de obtener resultados que sean agregados y corrijan errores en las predicciones. Se suele limitar el número de capas del árbol para asegurar que el algoritmo permanece débil.

Modelo aditivo Los árboles de decisión son agregados de manera individual, este cambio no influye en los modelos ya creados. Para la selección de parámetros de cada uno de los arboles de decisión agregados, se usa el Gradient Boost, que minimizará la función de pérdida. De este modo, podremos agregar árboles con distintos parámetros de manera que su combinación minimice la pérdida de nuestro modelo de predicción y mejora los valores predichos.

Este modelo es de los mejores dentro de los modelos de aprendizaje supervisado. El inconveniente principal es que requiere de un ajuste muy preciso y puede tardar bastante en llevar a cabo la fase de entrenamiento del modelo.

La Figura Figura 4.3 ilustra como este modelo realiza la predicción. En la primera fila podemos ver la primera interacción, en la cual observamos la columna de la izquierda correspondiente a los valores residuales y predicciones del árbol y la derecha correspondiente a la predicción de nuestro ensamble. Podemos observar que esta primera iteración ambos resultados son idénticos. En la segunda fila se realiza una nueva fase de entrenamiento la cual se basa en los en los errores residuales del primer árbol. Podemos observar que las predicciones del ensamble son iguales a la suma de las predicciones de los dos arboles anteriores. Un proceso similar se realiza para la tercera fila, podemos observar que las predicciones realizadas en nuestro ensamble van mejorando de manera gradual según se añaden mas arboles al ensamble.

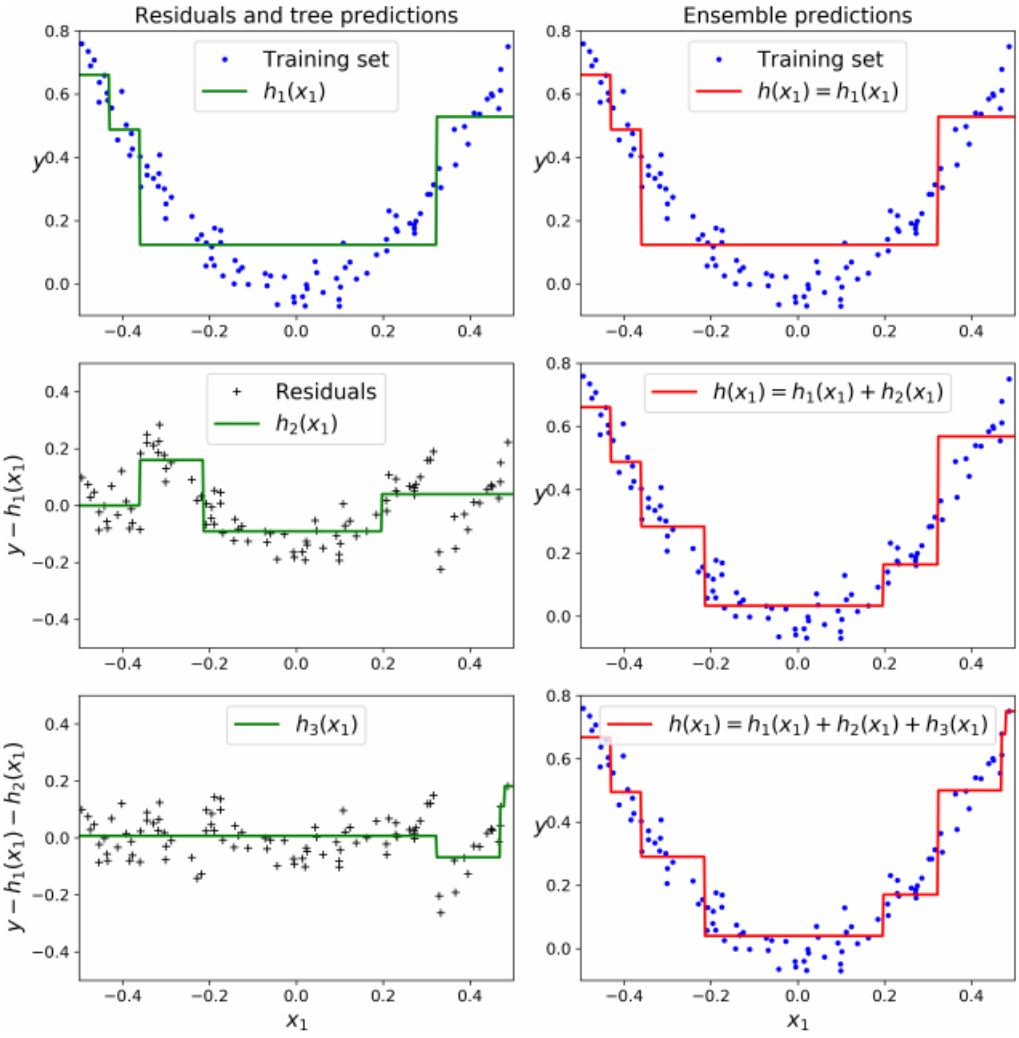


Figura 4.3: Funcionamiento de Gradient Boost [22]

4.2.3. Redes Neuronales - LSTM

En esta sección vamos a tratar de detallar como funciona una red neuronal, explicando posteriormente el funcionamiento de las redes recurrentes, hasta llegar a las redes LSTM, que son las que hemos usado para este estudio.

Redes Neuronales. Conocidas también como redes neuronales artificiales, son modelos computacionales inspirados en las redes neuronales biológicas. Estas son conectadas entre si para transmitir la información de entrada que posteriormente será sometida a una serie de operaciones que modifican el valor del resultado. Como resultado devolverán unos valores de salida.

Las redes neuronales son un conjunto de neuronas artificiales que usan una serie de dendritas para recibir la información del exterior o de otras neuronas. Cada una de estas conexiones tiene un valor y un peso. La neurona se encarga de realizar una función sumatorio para realizar el calculo de las entradas. Una vez tenemos el valor procedente de ese calculo, pasara por una función de activación. Las mas comunes son:

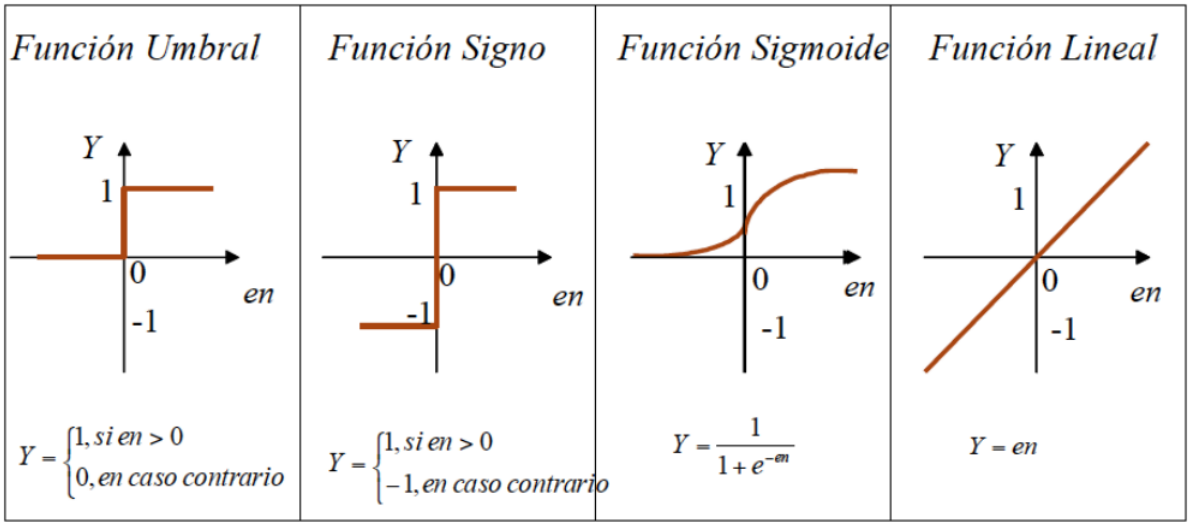


Figura 4.4: Funciones de activación [7]

Su función sera la de limitar la amplitud de la salida de nuestra neurona en función de su valor da salida. La neurona con la estructura mas sencilla es la denominada Perceptron, sus características son la entrada y salida en forma binaria y que su función de activación es la de umbral. La estructura que tiene esta es la siguiente:

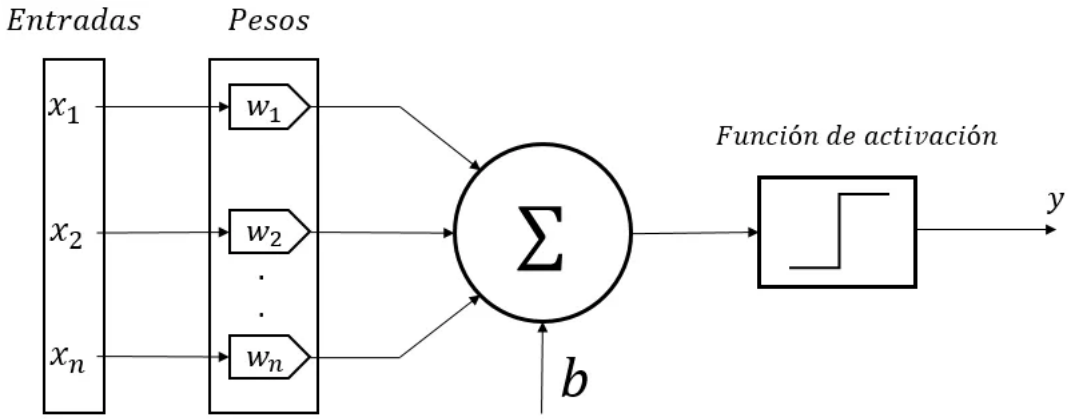


Figura 4.5: Estructura Perceptron

Redes Neuronales Recurrentes. Antes de hablar sobre las redes neuronales LSTM debemos hablar sobre las redes neuronales recurrentes, ya que las LSTM pertenecen a esta familia. Lo particular de estas redes neuronales es que son capaces de almacenar información a lo largo del tiempo. Juergen Schimdhuber [27] uno de los principales investigadores en este campo las define como:

"Las redes neuronales recurrentes permiten tanto la computación paralela como secuencial, en principio cualquier ordenador normal debería ser capaz de ejecutarlas, con diferencia en los tiempos de operaciones". Las Redes Neuronales Recurrentes son similares al cerebro humano, son de ingesta continua, por tanto pueden gestionar datos constantemente, se conectan a neuronas,que de algún modo son capaces de traducir entradas sensoriales de larga duración en una secuencia de salidas. La diferencia es que el cerebro es capaz de adoptar el rol de un modelo capaz de resolver varios problemas mientras que las máquinas no.

Antes estas redes eran difíciles de entrenar, pero con los recientes avances en optimización, arquitecturas de redes, paralelismo y el avance de las GPU's incorporadas para el entrenamiento de estas, su uso se ha facilitado.

En cuanto a cómo funcionan estas, la Red Neuronal Recurrente usa cada vector de una secuencia de vectores entrada y las modeliza a lo largo del tiempo. Esta técnica permite a la red retener el estado mientras se modelizan cada uno de los vectores a lo largo del conjunto de vectores de entrada (ver Figura 4.6):

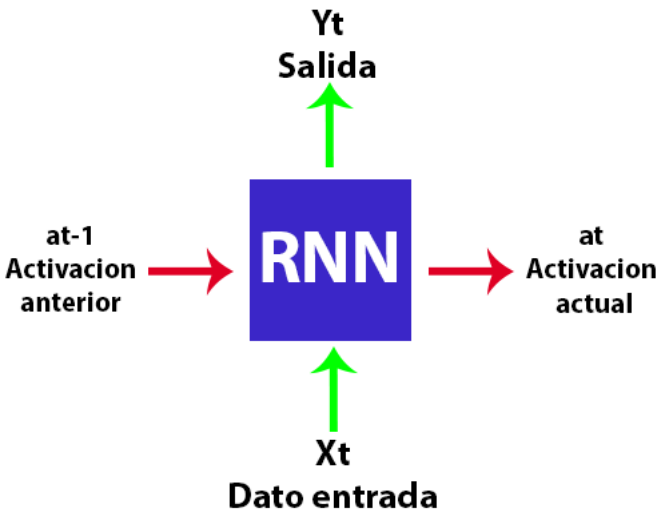


Figura 4.6: Modelización Red Neuronal Recurrente.

Es precisamente estas activaciones las que definen el termino de estado oculto, ya que permiten preservar y compartir la información entre un instante de tiempo y otro. Como podemos ver en la Figura 4.7 un ejemplo de vector normal y uno procedente de una red neuronal recurrente.

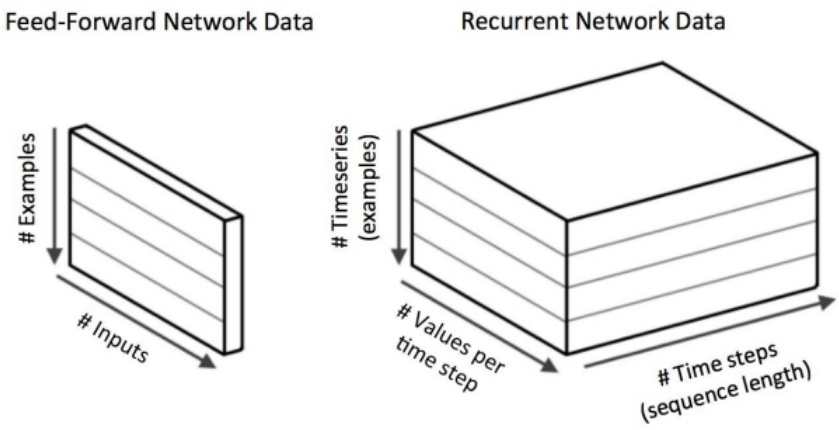


Figura 4.7: Comparación entre un vector de entrada normal y uno de Red Neuronal Recurrente.

En la subfigura de la izquierda, podemos ver que los datos se dividen en dos apartados. De manera vertical tenemos cada ejemplo y de manera horizontal tenemos las entradas que nos proporciona cada elemento.

En el modelo de la derecha, encontramos ciertas similitudes. De manera vertical seguimos teniendo los ejemplos pero estos son clasificados como serie temporal. En horizontal tenemos los valores que tiene cada serie con sus características. Y la otra dimensión

horizontal (*Time Steps*) contiene la longitud de la secuencia temporal que estemos almacenando.

Sin embargo estas redes tienen un problema denominado desvanecimiento de gradiente. Consiste en que los valores del gradiente son cada vez más pequeños a medida que el algoritmo progresa hacia las capas inferiores. A raíz de esto, la actualización del Descenso de Gradiente deja los pesos de las capas inferiores prácticamente sin cambios, y el entrenamiento nunca converge en una buena solución [22].

Redes Neuronales LSTM. Son una variación de las Redes Neuronales Recurrentes. Las LSTM fueron creadas en 1997 por Hochreiter y Schmidhuber [23].

Los componentes más destacables dentro de las LSTM son:

- Puertas de Actualización: Permite añadir elementos a la memoria.
- Puertas de olvido: Permite eliminar elementos de la memoria.
- Puerta de salida: Permite crear el estado oculto de la memoria actualizado.
- Celda de estado: Es una de las claves de las redes LSTM. Es como un camino por le cual pasa la información y podemos decidir si queremos añadir datos o eliminar datos de la memoria de la red. Para este propósito usamos las puertas citadas anteriormente.

Estos componentes son en si son redes neuronales que funcionan como arboles que permiten el paso y bloqueo de información. Estas puertas están formadas por 3 elementos:

- Red neuronal
- Función de sigmoide.
- Elemento multiplicador

El contenido de las células de memoria son modeladas por las puertas de entrada y las puertas de olvido. Asumiendo que ambas puertas permanecen cerradas el contenido permanecerá intacto entre los diferentes pasos que se darán.

La estructura en forma de puertas permite perpetuar la información a lo largo de los saltos temporales estos son periodos en los que no hay datos, por lo que la serie temporal presenta "vacíos" en los que el gradiente desaparece o se vuelve muy pequeño, y bloquea la incorporación de nuevos datos de entrenamiento. En consecuencia permite a la gradiente fluir por los diferentes pasos, esto permite al modelo LSTM superar el problema de desvanecimiento de gradiente.

Propiedades de las LSTM

Las propiedades que mejoran las LSTM son:

- Función de actualización mejoradas, estas ajustan los pesos en cada ciclo y en función de los pesos anteriores.
- Mejor propagación de la información a lo largo de la red, esto se produce gracias a las puertas de las que disponen estos modelos que les permiten olvidar o añadir información.

A continuación expondremos una serie de escenarios donde las LSTM funcionan muy bien [22]:

1. Generación de texto.
2. Clasificación de Series temporales
3. Reconocimiento del habla
4. Reconocimiento de la escritura
5. Modelado de musica polifónica

Arquitectura de LSTM. La arquitectura de una LSTM es bastante compleja, como vemos en Figura 4.8 ya que dispone de bastantes entradas y conexiones entre capas.

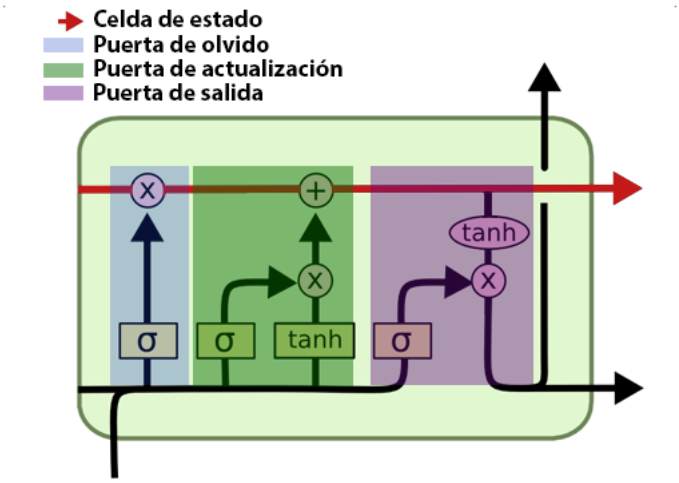


Figura 4.8: Arquitectura de celda LSTM

Esta arquitectura consta de una serie de entradas que son recogidas por nuestra capa de entrada denominada capa de entrada, las capas ocultas 1/2 son como una caja negra no sabemos lo que ocurre ahí, y después obtenemos la salida a través de la capa de salida.

Funcionamiento. Su composición interna tiene una línea base que va horizontalmente posicionada (Celda de estado) y a ella se agregan ciertas puertas mencionadas anteriormente, que son funciones que se activan y desactivan en función del criterio elegido para la función de activación, estas sirven para calibrar la red y su valor de salida oscila dependiendo de la función seleccionada.

Un factor importante de estas redes es que se haga la elección correcta de las funciones, que podemos observar dentro de las células y es fundamental el entendimiento de su estructura para obtener el mejor resultado posible.

Como hemos explicado antes, esta red es capaz de almacenar estados pasados. Para visualizar mejor en la Figura 4.9 podemos ver representados 3 diferentes estados de la misma red pero en diferente instante de tiempo.

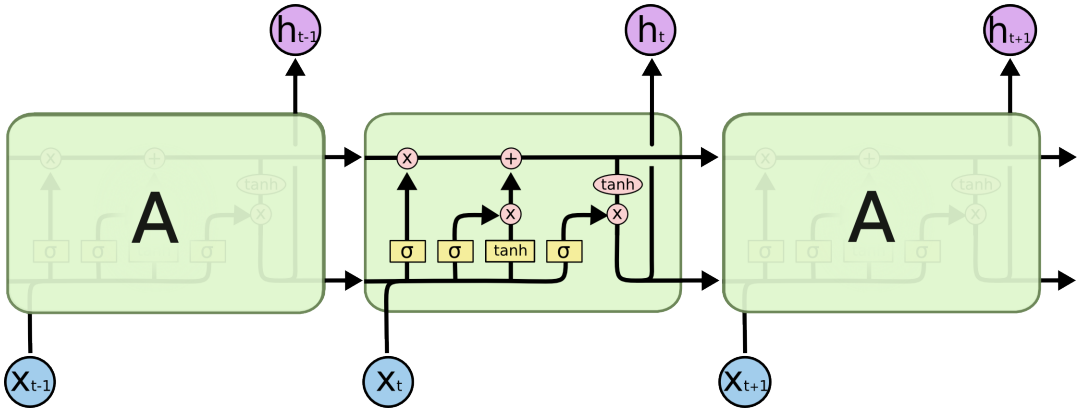


Figura 4.9: Varias neuronas LSTM

Capítulo 5

Bases de datos para series temporales

En este capitulo definiremos lo que son las bases de datos especializadas en series temporales. Estudiaremos sus características principales. Realizaremos un análisis de los mejores productos en esta categoría, exponiendo sus ventajas y desventajas. Posteriormente realizaremos un análisis en profundidad de las dos mejores, con el fin de elegir la que mejor se ajusta a nuestro problema.

5.1. Importancia de las bases de datos

Las bases de datos de series temporales surgieron, principalmente, enfocadas al entorno de la bolsa, como se ha mencionado anteriormente, aunque rápidamente se incorporaron más aplicaciones. Hoy por hoy no existe casi ningún sector al cual no se puedan aplicar de alguna manera series temporales, ya que hoy en día todo puede ser un componente de una aplicación, y formar parte de una amplia red de dispositivos capaces de recibir y tomar mediciones mediante sensores como pueden ser los teléfonos, coches o las calles de una ciudad.

Esto tiene una consecuencia, la generación de datos ha aumentado drásticamente y se hace mucho más difícil almacenar y operar con ellos. Cada vez hay más fuentes de datos, más herramientas, y justo ahí es donde más destacan las bases de datos de series temporales, ya que estas cuentan con una arquitectura y estructura capaz de escalar de forma rápida y hacer ingesta de todas las fuentes de datos sin tener dificultades de procesamiento.

Diferencia entre ACID y BASE. ACID son las siglas de (*Atomicity, Consistency, Isolation, Durability*). Para que una base de datos cumpla este paradigma debe procurar que se cumplan estas premisas.

- Atomicidad: Cuando una operación tiene varios pasos, si surge algún evento que interrumpe la acción en un paso intermedio hará o bien que se ejecuten todos los pasos o que no se cumpla ninguno, asegura, que las transacciones se ejecuten completas, para mantener, la consistencia de la base de datos.

- Consistencia: Esta propiedad asegura que las operaciones que se ejecutan sobre la base de datos sean operaciones válidas que no rompen las reglas de integridad, por tanto se asegura de llevar la base de datos de un estado válido a otro que también sea válido.
- Aislamiento: Esta es una propiedad que asegura que una operación no pueda afectar a otras operaciones, existen diferentes niveles de aislamiento ya que algunas aplicaciones de este pueden no necesitar un grado de restricción tan grande como otras.
- Durabilidad: Este propiedad hace que una operación persista y no pueda ser deshecha aunque nuestro sistema falle en algún momento. De esta manera se asegura que nuestros datos permanezcan inalterados.

BASE proviene de las siglas (*Basic Availability, Soft-state, Eventual consistency*) [1] como hemos explicado antes se diferencia bastante del modelo ACID.

- Disponibilidad básica: Esta propiedad hace que los datos estén operativos el mayor tiempo posible, permitiendo así un servicio ininterrumpido. Esto se produce gracias a que estos se almacenan de manera distribuida y cuentan con un alto grado de replicación. Así en caso de que surja algún tipo de fallo, no necesariamente debe interrumpir el servicio [2].
- Estado suavizado: Esta propiedad dictamina que las escrituras no tienen por qué ser consistentes, ni tampoco las réplicas existentes tienen que ser consistentes, esto ahorra mucho tiempo y brinda de mayor velocidad a bases de datos que usan este modelo.
- Consistencia eventual: Esto significa que el requisito que tienen estas en cuanto a coherencia es exigir que en algún momento del futuro los datos converjan a un estado que sea coherente. Pero no contamos con garantías de que vaya a ocurrir, ni cuando ocurrirá. Esto es algo totalmente diferente a las ACID que no admite una transacción hasta que no se haya completado la anterior y la base de datos haya pasado a un estado consistente.

5.2. Características

Una base de datos de series temporales se puede definir como un mecanismo software que está optimizado para la utilización de datos expresados en una franja de tiempo, como puede ser un intervalo de una fecha o hora. Estas tienen las siguientes características:

1. Concurrencia: Permite soportar altos picos de consultas y escrituras de manera simultánea, aunque lo más frecuente es que la mayor parte del trabajo de estas sea de escritura [29], ya que están pensadas para la ingesta de una gran cantidad de datos.

2. Consultas optimizadas: Las consultas en este tipo de bases de datos están optimizadas para realizar análisis de datos, llegando a poder realizar consultas de varios tera-bytes de datos en cuestión de segundos.

3. Orientadas al almacenamiento: Una de las características principales de este tipo de bases de datos es que son capaces de almacenar grandes cantidades de datos que pueden llegar hasta peta-bytes de información.

4. Alta disponibilidad: Cumplen con las características de una aplicación de alta disponibilidad proporcionando mecanismos para que la información esté siempre accesible.

5. Arquitectura distribuida: La razón principal por la que estas deben ser distribuidas es por los altos requisitos que tiene de escritura y almacenamiento [29]. Permite asegurar la alta capacidad de ingesta y concurrencia de consultas, facilitando así, servicios altamente escalables y de alta disponibilidad.

5.2.1. Características del modelo de datos

Las bases de datos de series temporales usan la marca de tiempo (*timestamp*) que está definida por el momento temporal en que fue tomada la métrica en cuestión. Esta puede tener varios parámetros adicionales que pueden aportar información sobre la marca. Los datos están organizados de la siguiente manera:

1. Marca temporal: Momento en el que se produjo la extracción de la métrica. En este tipo de bases de datos este campo funciona como clave primaria ya que nos servirá para realizar posteriormente búsquedas en rangos temporales de manera muy optimizada. Puede expresarse en diferentes formatos en función del nivel de precisión que necesitemos para cada aplicación a nuestros problemas, puede ser de una típica fecha a muestras en nanosegundos.

2. Origen: Lugar de donde proviene la métrica. Puede disponer de varias fuentes por tanto podemos indicar como ejemplo aplicado a nuestro problema, el avión del que se extrajo la métrica y en qué antena ADS-B fue recogida dicha métrica.

3. Medidas o mediciones: Se puede definir como los datos extraídos de la métrica, estas siempre van acompañadas de su marca temporal.

En la Figura 5.1 podemos ver un ejemplo de registro en una estructura de base de datos de series temporales.

timestamp	id	latitude	longitude	altitude	speed	vspeed	quawk	track	ground	leg
1517587275	fr24-34530E-1517587275	44.224740000000004	9.47696	26725	462	-960	6463	78	False	ANE8786_34530E_1517580840_15175

Figura 5.1: Modelo de datos

5.2.2. Ventajas y desventajas.

A continuación se van a exponer los puntos fuertes de la utilización de bases de datos de series temporales y los cuales han hecho que nos decantemos por este tipo de base de datos para dar solución a la problemática del almacenamiento de los datos.

Ventajas del uso de bases de datos de series temporales En el siguiente apartado vamos a exponer las ventajas del uso de bases de datos enfocadas a las series temporales.

1. Escalabilidad: La primera razón para elegir este tipo de bases de datos es por su gran capacidad de adaptación al aumento de los datos. Las bases de datos relacionales no soportan ciertos problemas que requieren almacenar una gran cantidad de datos: por ejemplo, millones de datos recogidos por un sensor y estos multiplicados por los distintos sensores disponibles. La explicación para que esto ocurra se reduce a que las bases de datos relacionales usan el paradigma ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad) mientras que las no relacionales por lo general siguen el paradigma BASE (Disponibilidad básica, Estado suavizado, Consistencia eventual). Este paradigma permite mayor ingesta de datos y fácil operatividad con ellos, mientras que los basados en el sistema ACID necesitan más garantías y pautas para asegurar los datos, haciendo así, que sean menos eficientes para tareas que necesiten alta escalabilidad. Para entender mejor el concepto vamos a definir más en profundidad las propiedades de cada uno de estos paradigmas.

2. Rendimiento: Si vamos a realizar búsquedas por marcas temporales (fecha, hora) es donde más brilla una base de datos de series temporales ya que las búsquedas sobre las marcas temporales están muy optimizadas, mientras que si se realiza la misma operación en otro tipo de base de datos puede ser un proceso muy costoso cuando se trata de datos que pueden superar los varios millones de registros.

Desventajas del uso de bases de datos de series temporales. Son varias las desventajas de usar una base de datos de este estilo, vamos a verlas a continuación explicadas de forma detallada:

1. Pérdida de datos: Es bastante común en sistemas de estas magnitudes y bajo las propiedades de los modelos BASE, que a veces con una ingesta tan grande de datos haya pérdidas. Dado el gran volumen de datos y su objetivo, el cual es la observación de tendencias, la perdida de registros individuales o pequeños rangos no suele ser un problema.

2. Problemas de adecuación con otras arquitecturas: Las bases de datos de este estilo tienen muchas particularidades sobre todo a la hora de recoger dichos datos y almacenarlos en forma de puntos. Este modo de almacenamiento puede dar problemas a la hora de adaptarlo a otra arquitectura o prestar servicio a otras aplicaciones.

3. Tiempo: Sin duda un factor muy determinante para montar una arquitectura basada en una base de datos de series temporales: se exige más tiempo, no para la ingesta de datos ya que es donde más liviana resulta esta tarea, sino a la hora de almacenar estos datos de un modo concreto o adaptarlos para su consumo y montar la arquitectura. Muchas de estas bases de datos disponen de *stack* tecnológico propio el cual a veces es complicado de tratar, ya que se necesita un conocimiento de toda la arquitectura y entorno.

5.3. Análisis exploratorio de bases de datos.

En esta sección se van a recoger las bases de datos de series temporales más relevantes y prometedoras del entorno, y se realizará un primer análisis para decidir cuáles son las dos mejores en características y rendimiento, de acuerdo con los objetivos de nuestro proyecto. Para comenzar con el análisis vamos primero a ver el *ranking* a día 9 de octubre (según sitio web DB-Engines) [1] de bases de datos de este estilo, para guiarnos mejor y seleccionar las más relevantes.

RANK	DBMS	SCORE		
SEP 2019		SEP 2019	24 MOS ▲	12 MOS ▲
1	**InfluxDB**	**18.65**	**+10.17**	**+6.85**
2	Kdb+	5.5	+3.71	+1.63
3	Prometheus	3.45	+2.78	+1.86
4	Graphite	3.32	+0.77	+0.62
5	RRDtool	2.57	-0.49	+0.02
6	OpenTSDB	1.86	-0.03	+0.06
7	Druid	1.73	+0.75	+0.53
8	TimescaleDB	1.4	+1.40	+0.92
9	KairosDB	0.54	+0.04	+0.01
10	GridDB	0.48	+0.40	+0.42

23 Systems in Ranking. September 2019

Figura 5.2: Ranking bases de datos de series temporales [40]

En base a este *ranking* analizaremos estas bases de datos intentando definir cuál se puede ajustar más a nuestros casos de uso y objetivos. Este primer análisis es exploratorio, con el propósito de elegir las dos mejores y, sobre ellas realizar otro análisis en mayor profundidad.

[1] `https://db-engines.com`

Se procederá al análisis en orden descendente a la tabla de *ranking* comenzando con la primera en figurar en la lista que es InfluxDB.

1. InfluxDB: Es una base de datos de alto rendimiento desarrollada especialmente para datos con marcas temporales. Esta nos permite la ingesta de gran volumen de datos y procesamiento con consultas en tiempo real. Está desarrollada completamente en el lenguaje de programación Go, no presenta dependencias externas. Permite la creación de dashboards interactivos unos paneles que permiten operar y visualizar contenido de manera personalizada mediante consultas. Usa el *stack* tecnológico denominado como TICK (*Telegraph, InfluxDB, Chronograph, Kapacitor*). Cabe destacar que esta es open source con licencia MIT [2] por tanto se puede usar por todo el mundo de manera gratuita. Si se necesita asistencia técnica personalizada para implantar este *stack* tecnológico dentro de nuestra empresa el servicio es de pago.

El lenguaje de consultas es parecido al lenguaje SQL, y además dispone de apis tanto HTTP API como JSON. Soporta una gran variedad de lenguajes de programación (.Net, Clojure, Erlang, Go,Haskell, Java, JavaScript, JavaScript, Node.js, Lisp, Perl, PHP, Python, R, Ruby, Rust, Scala). Soporta particionado horizontal, en la versión enterprise, lo que nos permite distribuir la base de datos en diferentes máquinas conservando su esquema. La versión enterprise también soporta niveles de replicación, esto significa que se puede tener más de una copia de las filas en diferentes nodos.

Soporta consultas concurrentes, como hemos dicho antes, una característica fundamental dentro de las bases de datos de series temporales. Esta base de datos puede ser usada en memoria, proporcionando aun más velocidad a las consultas.

El modelo de datos usado por InfluxDB es el modelo de datos columnar, muy común dentro de las bases de datos NoSQL. Para la indexación hace uso de diferentes tipos de técnicas de indexación. Al tratarse de una base de datos con características diferentes a las relacionales, esta no soporta claves foráneas ni *join* de tablas. Dispone de varios métodos de compresión que se adaptan en función de los datos almacenados. InfluxDB dispone de algo conocido como consulta continua, que es conceptualmente parecido a las vistas materializadas.

2. Kdb+: Es una base de datos columnar y relacional orientada a series temporales. Sus consultas se realizan en un lenguaje parecido a SQL llamado q. Tiene un amplio soporte de api's HTTP API, Jupyter, Kafka, WebSocket, JDBC, ODBC. Los lenguajes de programación soportados por esta base de datos son los siguientes (C, C#, C++, Go, J ,Java , JavaScript, Lua, MatLab, Perl, PHP, Python, R, Scala). Cabe destacar que esta base de datos tiene licencia comercial, por tanto su uso no es gratuito.

Soporta consultas concurrentes. La particularidad de esta es que usa particionado de las consultas en función de su marca temporal, esto significa que ordena las consultas y las ejecuta en función de su marca temporal. El modelo de datos usado por esta base de datos es el modelo relacional. Existe una problemática en el uso de modelos relacionales

[2] https://es.wikipedia.org/wiki/Licencia_MIT

aplicados a series temporales, este proviene de la gran cantidad de datos que se generan con las series temporales y la dificultad de mantener una estructura de almacenamiento relacional con tantos datos. Para remediar esto, Kdb+ realiza compresión de los datos en disco y después los distribuye en diferentes maquinas con el fin de reducir la carga de volumen de datos.

Al ser un modelo relacional también soporta claves foráneas. Tiene soporte para la utilización de índices, tanto primarios como secundarios. Dentro de los diferentes niveles de aislamiento, esta solo soporta el nivel de aislamiento serializable: cuando una transacción está en el nivel serializable, la consulta sólo ve los datos cursados antes de que la transacción comience y nunca ve ni datos sucios ni los cambios de transacciones concurrentes cursados durante la ejecución de la transacción, gracias a que las consultas se ejecutan en función de una marca temporal. Esta base de datos también permite el uso de operaciones join.

Dispone de soporte tanto para almacenar la información en disco como para almacenamiento en memoria, por tanto le permite ser más versátil en ese sentido.

3. Prometheus: Otra de las bases de datos de series temporales que se encuentran dentro del ranking de las más usadas. Está es desarrollada en el lenguaje de programación GO, y soporta Java, Ruby, .Net, C++, Haskell, Javascript y Python. Es un proyecto de código abierto. Utiliza un sistema de puntos de guardado que, por defecto, realiza guardados periódicos cada dos horas. Estos puntos se componen de una compresión de los logs de escritura con el tiempo más actualizado y si existe un punto de guardado anterior lo utiliza también. Se almacenan en un mismo directorio con notación numérica, asegurando la consistencia de la base de datos.

Prometheus permite el almacenamiento de tuplas formadas por una marca temporal y un valor numérico, por tanto existen varios tipos de compresión que puede llevar a cabo. Para la compresión de marcas temporales utiliza el algoritmo llamado delta of delta [38], consiste en aplicar compresión delta dos veces, esto consiste en partir los bytes de la primera marca temporal recogida avanzar en las siguientes y almacenar solo los valores alterados producidos por la diferencia. Como las marcas temporales están muy próximas entre si el cambio entre una marca y otra es mínimo, por lo que el valor que se comprime es muy pequeño, no un timestamp completo. No soporta la concurrencia.

Es compatible con la creación de índices, estos se forman generando listas de series que se crean mediante el uso de índices invertidos. Lo peculiar de estos índices es que a diferencia de los índices en SQL los cuales se definen a priori, en los invertidos el índice se crea a posteriori[3], esto ocurre una vez el motor haya analizado los documentos con los cuales vas a realizar la búsqueda. Y es gracias a eso que puede acceder más rápido a la información ya que no necesita recorrer todos los campos solo la lista correspondiente al valor buscado. Al no tratarse de una base de datos relacional no soporta Joins. La arquitectura de Prometheus nos permite almacenar nuestros datos de dos maneras posibles: de forma local, o mediante almacenamiento externo en Kafka, PostgreSQL, Amazon S3

[3]https://dbdb.io/db/prometheus

usando ProtoBuffer para adaptarlo a cada una de las diferentes opciones.

4. Graphite: Base de datos de código abierto con licencia Apache 2.0, y desarrollada en el lenguaje de programación Python. Es compatible con APIS HTTP y Sockets, soporta Python y JavaScript en concreto node.js. Dispone de consultas concurrentes. Esta base de datos tiene la peculiaridad de estar orientada al almacenamiento numérico de series temporales y es capaz de realizar renderización de gráficos con los datos almacenados[4], muy especializada en problemas que requieran de uso de grafos. Esta dispone de su propia arquitectura formada por tres módulos que son: Carbón servicio de ingesta de alto rendimiento capaz de obtener una gran cantidad de datos; Whisper el módulo encargado de guardar los datos provenientes de la ingesta de Carbon y Graphite-web que es la interfaz gráfica, esta nos permite realizar módulos para visualizar los grafos.

Se puede realizar partición de los datos mediante la técnica de *consistent hashing* (hash consistente): que permite distribuir el esquema de la base de datos distribuida en varios servidores. El hash consistente es un tipo especial de hash, de tal manera que cuando se cambia el tamaño de una tabla de hash, sólo se necesita rehacer las claves del estilo n/m en promedio, donde n es el número de claves y m es el número de *sockets* (ranuras).

Este sistema es compatible con los servicios como Amazon Web Services CloudWatch con el módulo de Influx Grafana. Su principal objetivo es visualización en tiempo real de datos.

Permite operaciones concurrentes mediante el protocolo basado en bloqueos que permite 4 posibles escenarios:

- Bloqueo compartido: También llamado solo lectura, los datos pueden ser compartidos entre las transacciones, no dispondremos permisos de modificación sobre los datos y así no podremos generar inconsistencias dentro de la base de datos.
- Bloqueo exclusivo: Puede tanto leer como escribir los datos con los que se opera, por esto no es compatible con el uso de transacciones concurrentes.
- Bloqueo simple: Permite realizar un bloqueo sobre algún dato antes de empezar a operar con el evitando así que otra transacción realice un bloqueo sobre ese bloque de datos.
- Bloqueo pre-petición: Este permite evaluar la transacción que se va a realizar, crear una lista de requisitos para que se ejecute y no iniciarla hasta que todos los requisitos sean cumplidos, una vez se cumplen se bloquea el bloque necesario y se realizan las operaciones.

5. RRDtool: Las siglas de esta base de datos provienen de Round-Robin Database, debido a que el buffer de la base de datos está almacenado de manera circular [21]. Tiene una licencia de uso de código abierto, en concreto GPL V2. La particularidad de esta base

[4]https://graphite.readthedocs.io/en/latest/faq.html (2019-17-8)

de datos es que esta orientada a la nube, y por tanto no esta disponible para el uso local. Está desarrollada en el lenguaje de programación C.

Por defecto, los tipos de datos a los que está orientada esta base de datos son datos numéricos o fechas. Aparte de C, soporta, los siguientes lenguajes de programacion: C#, Java, JavaScript en concreto Node.js, Lua, Perl, PHP, Python y Ruby.

Al no tratarse de una base de datos relacional, no es compatible con claves foráneas, y tampoco dispone de medios para asegurar la consistencia bajo entornos distribuidos.

Sí que soporta concurrencia de manejo de datos mediante el uso de *rrdcached deamon*. El funcionamiento de este mecanismo se resume en que es un proceso demonio ejecutado que recibe las modificaciones que se han realizado sobre algún fichero con extensión (.rrd) donde se guardan los datos. En cuanto haya un número suficiente de cambios o haya pasado un tiempo predeterminado, escribe los cambios en los archivos mencionados anteriormente.

6. OpenTSDB: Otra base de datos dentro del ranking con la peculiaridad de estar implementada en el lenguaje de programación Java. Aparte de este, los lenguajes que soporta también son Erlang, Go, Python, R y Ruby . Además, es compatible con APIs como HTTP y Telnet.

Dispone de particionado para almacenar la informacion de manera distribuida en diferentes nodos. La técnica que usa es el denominado Sharding [24], y que consiste en particionar la base de datos en "fragmentos"más pequeños que son fácilmente almacenables. También soporta replicación, por tanto la información se puede encontrar almacenada de forma redundante en varios nodos.

Dispone de un mecanismo que asegura la consistencia de los datos, *Inmediate Consistency* o consistencia inmediata, que consiste en validar continuamente la consistencia de los datos con cada procesamiento de estos. En cuanto a la atomicidad usa un tipo de operaciones conocidas como todo o nada: o se ejecuta toda la transacción de manera correcta o no se ejecuta.

7. Druid: Está centrada en la analítica de datos ya que esta diseñada para ejecutar sentencias bajo procesamiento analítico en linea (*OLAP*) [5]. Para cumplir con este propósito, hace acopio de estructuras multidimensionales que contienen un resumen de gran cantidad de datos. Por tanto, donde más destaca esta base de datos es en las consultas que tengan que ver con los SELECT. Está enfocada principalmente en el campo de Buissnes Inteligence.

Es una base de datos de código abierto bajo licencia Apache v2, esta desarrollada en el lenguaje de programación Java aunque también es compatible con otros lenguajes como Clojure, JavaScript, PHP, Python, R, Ruby o Scala. No dispone de soluciones en la nube, y permite índices secundarios. Es compatible con API's externas como JSON. Permite particionado, con la técnica de fragmentación o Sharding. Permite almacenamiento redundante en múltiples nodos vía HDFS, S3 (Servicio de Amazon).

Dispone de métodos que garantizan la consistencia de la base de datos, en particular esta usa el método de la consistencia inmediata, explicada anteriormente en otras bases de datos. Permite operaciones de manera concurrente.

8. TimescaleDB: una base de datos optimizada para una rápida ingesta de datos y visualización de consultas complejas. Esta basada en PostgreSQL. No es exclusiva de series temporales, también acepta el modelo relacional. Es de código abierto bajo licencia de Apache V2, está implementada en el lenguaje de programación C aunque soporta los siguientes lenguajes .Net, C++, Delphi, Java, JavaScript, Perl, PHP, Python, R, Ruby, Scheme, Tcl. Tiene alta compatibilidad con API's externas como ADO.NET, JDBC, ODBC.

Al ser una base de datos que soporta el modelo relacional, acepta claves foráneas y también dispone de índices secundarios. Como peculiaridad esta base de datos si soporta los triggers. También esta disponible la opción de realizar joins.

Permite el particionado de los datos en diferentes nodos mediante particionado hash. También usa métodos de replicación de datos en múltiples nodos con el fin de estar disponibles si hay fallos. En particular, esta base de datos usa el método de maestro-esclavo mediante escritura en caliente y lectura en los esclavos.

Como otras bases de datos para el control de consistencia hace uso de la consistencia inmediata: las transacciones tienen que cumplir con el estándar ACID que hemos explicado en el anterior apartado. Al cumplir con el estándar ACID eso significa que también cumple con los niveles de aislamiento de las consultas (*Read Uncomitted, Read Committed, Serializable, Repeteable Read*)

Ante la gran cantidad de bases de datos de series temporales que existen y con el fin de poder visualizar de manera más fácil qué funciones dispone cada una de ellas se ha realizado una tabla (ver Figura 5.3) con el fin de sintetizar la información aportada anteriormente:

Características	InfluxDB	Kdb+	Prometheus	Graphite	RRDtool	OpenTSDB	Druid	TimescaleDB
Modelo de la base de datos	Series temporales	Relacional	Series temporales	Series temporales	Series temporales	Series temporales	Relacional/Series temporales	Series temporales
Licencia	MIT	Comercial	Apache2.0	Apache2.0	GPL V2	LGPL	Apache2.0	Apache2.0
Esquema de datos	Sin esquema	Si	Si	Si	Si	Sin esquema	Si, también soporta sin esquema.	Si
Triggers	No	Si	No	No	No	No	No	Si
Claves Foráneas	No	Si	No	No	No	No	No	Si
Concurrencia	Si	Si	Si	Si	Si	Si	Si	Si
Consistencia	-	Consistencia inmediata	No	No	No	Consistencia inmediata	Consistencia inmediata	Consistencia inmediata
Replicación	Factor de replicación.	Maestro-Esclavo	Si	No	No	Factor de replicación seleccionable.	Si, HDFS, S3	Maestro-Esclavo
Capacidad de uso en memoria	Si	Si	No	-	Si	No	No	No
Lenguaje de desarrollo	Go	q	Go	Python	C	Java	Java	C
APIs	HTTP API JSON con UDP	HTTP API Jupyter Kafka WebSocket JDBC ODBC	RESTful HTTP/JSON API	HTTP API Sockets	-		JSON con HTTP	Streaming C API ADO.NET JDBC ODBC

Figura 5.3: Tabla comparativa bases de datos.

5.4. Comparativa Influx vs Prometheus

A continuación se realizará una comparación en profundidad de estas bases de datos, en base al estudio previo realizado de las múltiples bases de datos. Estas dos son las que mejor se ajustan por características a las necesidades del problema, y además son las que mejor rendimiento nos proporcionarán a priori.

Para realizar de manera detallada la comparativa vamos a estudiar diferentes aspectos de la misma que nos permitan ver cual de las dos bases de datos que estamos comparando podrá darnos un mejor rendimiento y mejor experiencia de uso.

5.4.1. Estructura

A continuación vamos a realizar un análisis de la estructura de cada una de ellas. InfluxDB tiene la particularidad de que es una base de datos sin esquema (*schemaless*). Esto quiere decir, que a la hora de almacenar datos en ella no hace falta que se declare el tipo de dato que se va a almacenar o cómo se va a almacenar el dato. Incluso se pueden agrupar por datos que no tienen la misma estructura a estos se les denomina *schema on write*. Normalmente en las bases de datos relacionales antes de realizar cualquier almacenamiento se necesita declarar un tipo de estructura fijo sobre el que va a almacenar los datos, al contrario que las *schemaless*.

En este caso InfluxDB y Prometheus se diferencian enormemente ya que Prometheus si necesita una estructura declarada explícitamente antes de la inserción de datos mientras que InfluxDB no. Esto se adapta mejor al rendimiento que queremos obtener gracias al enfoque schemaless de InfluxDB, ya que este permite de manera mas eficiente la ingesta de datos, y que no necesitaremos cumplir condiciones estructurales como en Prometheus.

5.4.2. Modelo de datos

Ambas tienen un modelo similar basado en puntos, la diferenciación es que en InfluxDB un punto se compone por cuatro componentes que son: la medida, un conjunto de etiquetas, campos y la marca temporal. El conjunto de etiquetas o tagset cumple la función de un diccionario de claves - valor, en los que el valor está representado como una cadena de caracteres, y la clave como el índice. Los campos es donde almacenamos todos los datos relacionados con la métrica tomada. La marca temporal es el momento en el que se ha tomado la métrica, y la medición indica a qué conjunto pertenece el punto. Por ejemplo: La medición diaria de temperatura, en esta existen puntos de los cuales se ha realizado una medición en diferentes intervalos horarios del día.

En Prometheus hay un grado de similitud en cuanto a que ambas almacenan métricas con claves - valor. Pero como comentamos antes, la estructura en este caso debe ser definida antes, de manera que le tenemos que indicar <métrica><atributo><valor>y el campo de la marca temporal que va aparte de la estructura. Prometheus dispone de una serie de funciones como son el Inc()/Des() que pueden detectar tendencias dentro de la serie temporal. Existen funciones para alterar la marca temporal, permite hacer consulta

en rangos de tiempo aunque esto es algo más común dentro de las bases de datos de series temporales. También cuenta con una función Observe() que nos permite visualizar información estadística sobre la serie, con esta ultima función también cuenta InfluxDB.

5.4.3. Índices

En ambas bases de datos se usan los llamados índices invertidos, estos son muy usados en motores de búsqueda. En el caso de este tipo de bases de datos, hace que las consultas sean mucho más rápidas, y basándonos en su estructura diseñada para el almacenamiento de datos a gran escala y rápido acceso a estos.

El funcionamiento del índice invertido [42] para texto se realiza de la siguiente manera, primero recoge la información y la tokeniza separa el documento en fragmentos más pequeños. Lo siguiente que hace es separar de manera lógica esas cadenas más pequeñas y que cumplan un mismo estándar. Una vez se han realizado estos pasos el siguiente paso es crear los índices con las cadenas ya normalizadas.

Aparte de este método InfluxDB usa otro más denominado LSM [34] (*Long structured merge-tree*) permiten crear índices de grandes volúmenes de datos como es el caso de los ficheros log. Los árboles LSM suelen tener dos estructuras diferenciadas y separadas que están optimizadas para su medio de almacenamiento, los datos se sincronizan entre estructuras por lotes.

Estos árboles se usan en varios niveles, pero lo que los hace particularmente más rápidos es que el primer nivel lo mantiene en memoria, y es que InfluxDB tiene capacidad de operar en memoria a diferencia de Prometheus que carece de esa característica. Para poner un ejemplo más visual de cómo funciona el árbol, tenemos que imaginar que cada nivel del árbol, cuanto más profundicemos en los niveles más difícil y costoso es obtener resultados. Para prevenir esto, cuando el árbol empieza a ser muy profundo se hace una fusión de generaciones y se crea una nueva generación en función de las dos antiguas.

5.4.4. Modelo de almacenamiento

En este apartado vamos a definir cómo es el modelo de almacenamiento de cada una de las bases de datos. InfluxDB usa el modelo de almacenamiento por descomposición columnar, esta hace acopio del *sharding*, visto anteriormente. Al descomponer los datos en pedazos que están descompuestos por la marca temporal permite facilitar la eliminación de los archivos una vez estos dejan de ser usados. Debido a esto no se hacen grandes actualizaciones en los datos para que estos pasen a ser persistentes. También es más fácil realizar consultas en determinados rangos de tiempo y a su vez aplicarle funciones como serían la media, mínimos, máximos etc...

En cuanto a Prometheus, está basado en un modelo de almacenamiento denominado N-Ary model. En este modelo todos las relaciones son almacenadas de manera conjunta, y se asume que la relación será N-aria (véase Figura 5.4). Esto dará lugar a un almacenamiento en forma de sucesión de N-tuplas, expresado en forma de tablas: Los datos se almacenaran fila por fila donde N sera el número de columnas en nuestra tabla.

Para nuestras necesidades es mejor tener un modelo columnar el cual nos permita el tratamiento e ingesta de gran cantidad de datos.

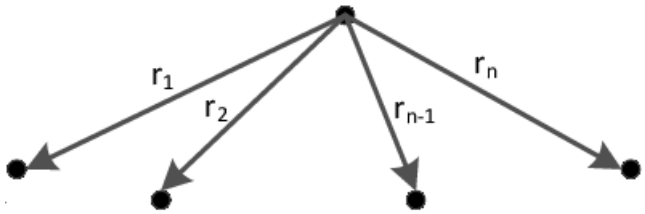


Figura 5.4: Esquema relación N-aria.

5.4.5. Compresión de los datos

En esta sección estudiaremos cómo se comportan las bases de datos seleccionadas con respecto a la compresión de los datos. InfluxDB dispone de varias técnicas de compresión, entre ellas están la codificación por diccionario, codificación de longitud de ejecución, compresión por empaquetado de bits. La selección de una técnica u otra varía en función de cómo son los datos.

- La codificación por diccionario [14]: pretende buscar coincidencias entre una cadena de caracteres y la estructura denominada diccionario con el fin de realizar una sustitución de la cadena por la posición donde se puede encontrar en el diccionario, realizando así, una compresión de los datos.
- Run-Length Encoding: simplemente sustituye los caracteres repetidos por el número de veces que se repite seguido del carácter en cuestión. La virtud que tiene es que es un sistema muy sencillo para la compresión numérica, como hemos dicho antes depende mucho de los tipos de datos ya que este método no es bueno con largas cadenas de valores ya que para codificar un numero de 8 cifras seria necesario una cantidad de 32bits. A continuación se muestra un ejemplo de codificación mediante este método:

```
Entrada:
    BBBBBBBBBBBBNBBBBBBBBBBBBNNNBBBBBBBBBBBBBBBBBBBBBBN
Salida después de la compresión:
    12B1N12B3N24B1N
```

En cuanto a Prometheus usa un algoritmo denominado compresión delta [38] es una técnica convencional consiste en aprovechar la existencia de versiones de los datos individuales, codificando cada nueva versión como los bytes que cambian con respecto a la última versión disponible. De esta manera esto, un ahorro considerable de tamaño y se optimiza el uso de los recursos.

Considero que aplicados a la problemática actual, lo óptimo es el uso de las diversas técnicas de compresión que ofrece InfluxDB, bien es cierto que podremos hacer uso de la compresión delta que nos ofrece Prometheus, pero InfluxDB cubre mas escenarios con sus diversas técnicas de compresión.

5.5. Justificación elección BBDD

Para concluir con este punto, vamos a exponer las razones que nos han llevado a decantarnos por InfluxDB como base de datos para el despliegue de este proyecto. Aunque ha sido complicado decantarme por una u otra ya que tenían características bastante similares, los factores claves a la hora de elegir esta base de datos han sido los siguientes:

1. Capacidad de ingesta de datos: Sin duda uno de los puntos críticos, aunque en un primer planteamiento en este proyecto no se haga ingesta en tiempo real de los datos, sí se prevé que en un futuro proyecto su uso en producción sea con ingesta de datos en tiempo real, y por tanto debe ser capaz de poder manejar ese gran volumen de datos.

2. Rápida respuesta: Al tratarse de una base de datos exclusiva de series temporales InfluxDB es capaz por su estructura de manejar sentencias de visualización y obtención de información de manera mucho más rápida de lo que lo haría cualquier otra, y en este punto en particular es donde esta sin duda destaca más que ninguna otra. Su manera de almacenar los datos sin necesidad de definir una estructura al principio es una gran ventaja ya que puede hacer la ingesta desde varias fuentes diferentes y después combinarlos y modificarlos al gusto.

3. Escalabilidad: Cabe destacar que InfluxDB tiene la posibilidad de escalar de manera horizontal de forma gratuita. Si se necesita un escalado vertical (*clustering*) es necesario adquirir una edición Enterprise. Esta opción también nos brinda una mayor libertad ya que si a lo largo del proyecto la estructura de los datos cambia, InfluxDB se adaptará a los cambios, sin necesitar modificaciones previas.

4. Licencia: Al tratarse de una base de datos con una licencia *open source* MIT nos permite realizar soluciones sin que nos conlleve un coste. Es verdad que existen planes empresariales que se dedican a añadir soporte más especializado para ayudar a implantar InfluxDB a su sistema, pero si se dispone de tiempo se puede hacer sin ningún coste.

5. Soporte: Como hemos mencionado antes existe soporte general y soporte más especializado de pago, aunque el soporte gratuito cumple bastante más de lo esperado ya que hay muchos canales de comunicación directos con ellos y comunidades de desarrolladores que dan mucho feedback entre ellos.

6. Visión de futuro: InfluxDB es una base de datos joven, pero con una proyección enorme [5][6] dentro del sector de las bases de datos de series temporales. En pocos años ha conseguido posicionarse en lo más alto de los rankings de bases de datos de este tipo, por su eficacia, rapidez y sobre todo por las características de su arquitectura TICK que definimos más atrás en la memoria.

Teniendo en cuenta los puntos anteriormente citados, para el problema que se va a abordar se usará InfluxDB como base datos. Con el fin de poder realizar consultas en base a las marcas temporales, y así ganar precisión a la hora de seleccionar que datos queremos usar para alimentar el modelo de predicciones. En un principio los datos que se van a importar no van a cambiar, pero en un futuro a la hora de ponerse en producción. Este tipo de bases de datos permiten anticiparnos a futuros cambios en la estructura pudiendo realizar así cambios de manera más dinámica. Por otro lado la gran capacidad de escalar que tiene el *stack* arquitectónico TICK que en nuestro problema no será necesario ya que nuestros, datos no son excesivamente grandes, no obstante para futuras aplicaciones y puesta en producción se usaran grandes volúmenes de datos en los cuales sí será necesaria dicha capacidad de escalado.

[5]https://www.influxdata.com/blog/influxdata-raises-35m-to-accelerate-growth-and-meet-growing-global-demand-for-time-series-database/

[6]https://www.influxdata.com/blog/influxdata-secures-60-million-in-series-d-funding-to-bring-the-value-of-time-series-to-the-enterprise-mainstream/

Capítulo 6

Construcción del Modelo de Aprendizaje

En este capítulo vamos a hablar sobre el desarrollo de nuestro estudio, este se ha realizado usando el lenguaje de programación Python sobre Jupyter Notebook un interprete especializado en tareas analíticas. También explicaremos cómo gracias a los datos obtenidos por los receptores ADS-B, somos capaces mediante una serie de procesos de llegar a crear un modelo que sea capaz de predecir el momento de llegada de la aeronave al aeropuerto. Tendremos que comprender bien los datos para poder extraer información valiosa y que nuestro estudio tenga éxito.

Posteriormente, mostraremos los resultados obtenidos a lo largo de este proyecto de investigación, haciendo una reflexión sobre estos. Cabe destacar que el principal objetivo de este proyecto es precisamente el poder llegar a obtener resultados válidos, los cuales puedan servir para mostrar la capacidad que tienen estos modelos y su eficacia aplicada a problemas de este tipo.

6.1. Descripción del conjunto de datos

En esta sección describiremos los datos de los que disponemos. Partimos del *Dataset* este dispone de 1209742 filas y 16 columnas más una para el índice, un peso total de 213 MB. Es un archivo CSV, el separador entre datos es la coma. Las características que contiene son las siguientes:

Nombre característica	Tipo	Descripción
timestamp	Int	Indica la marca de tiempo en la que se realizó la medición.
latitude	Float	Indica la latitud de la aeronave en el momento de atención del dato.
longitude	Float	Indica la longitud de la aeronave en el momento de atención del dato.
altitude	Int	Indica la altitud de la aeronave en el momento de atención del dato.
speed	Int	Indica la velocidad de la aeronave en el momento de atención del dato.
vspeed	Int	Indica la velocidad vertical de la aeronave en el momento de atención del dato.
ground	Boolean	Si la aeronave esta tocando suelo o no.
leg	Str	Identificador de la aeronave.
flight_operator	Str	Operador al que pertenece la aeronave.
leg_tbegin	Int	Marca temporal comienzo de vuelo
leg_tend	Int	Marca temporal fin de vuelo
estimated_deaperture_time	Int	Tiempo estimado de hora de salida del hangar.
estimated_arrival_time	Int	Tiempo estimado de hora de llegada al hangar.
airport_origin	Srt	Aeropuerto origen
airport_destination	Str	Aeropuerto destino
aircraft_model	Str	Modelo de aeronave.

Figura 6.1: Características del Dataset

6.2. Preparación de los Datos

Antes de la creación de un modelo sólido, necesitamos realizar un estudio en profundidad de los datos que obtenemos de ADS-B y los datos ofrecidos por el aeropuerto de Madrid Barajas-Adolfo Suárez. Previamente hicimos un pequeño estudio de estos, ya que necesitábamos saber cómo iban a responder nuestros datos dentro de una base de datos de series temporales. Por eso antes de nada juntamos los dos conjuntos de datos, por un

lado tenemos los datos provenientes de los receptores ADS-B y por otro lado tenemos los datos proporcionados con información del aeropuerto. Mediante el identificador del vuelo podemos juntarlos en un solo conjunto de datos.

En primera instancia, nuestro modelo de datos tenía la forma descrita en la (Figura 2.5), pero posteriormente nos dimos cuenta que para llevar a cabo la predicción del tiempo de llegada del vuelo nos faltaba información sobre el despegue (*Takeoff*) que corresponde al último punto en el cual el avión tiene contacto con la pista, y el aterrizaje (*Touchdown*) que corresponde a cuando el avión hace el primer contacto con la pista, así sabremos que el avión ha tomado tierra.

Por ello, en primer lugar hemos visualizado cómo estaban dispuestas las series, con el fin de hacernos una idea de cómo obtener estos dos puntos. Estas son algunas de las visualizaciones que obtuvimos:

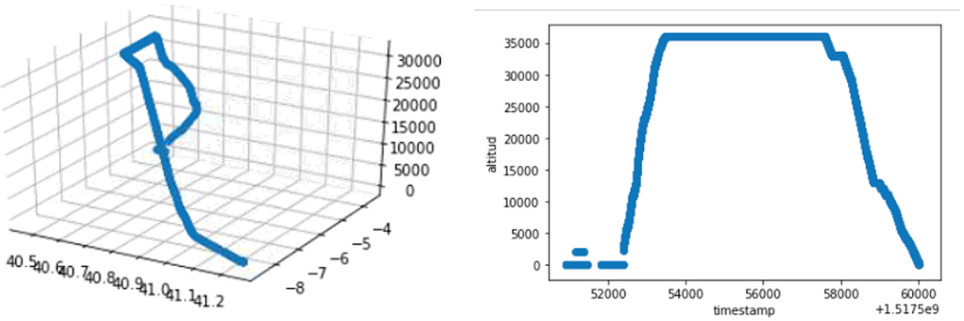


Figura 6.2: Visualización Serie 3D/2D

En ambas figuras podemos ver el perfil del vuelo. En la imagen de la izquierda podemos ver un gráfico tridimensional en el que podemos ver representados en sus ejes; x: Latitud, y: Longitud y z: Altitud. Así mismo en el gráfico de la derecha vemos ese mismo vuelo de manera bidimensional, representando en su eje x: periodo en el que se tomo la muestra, y en el eje y: altitud.

Tras visualizar varias series, nos dimos cuenta de que algunas tenían cortes al principio de la serie o al final. Esto podía ocasionar problemas a la hora de definir el punto de despegue y aterrizaje mencionados anteriormente. Por ello tuvimos que tomar una serie de medidas para asegurarnos de que dichos puntos se elegían de manera correcta. Tras realizar una serie de pruebas obtuvimos los puntos del siguiente modo:

- **Takeoff:** para hallar este valor lo que hemos realizado es, en primer lugar ordenar los mensajes de cada vuelo de manera cronológica. Después se identifica el primer mensaje del avión en el que este se encuentre en el aire (Ground = False). A continuación identificamos el valor del ultimo mensaje en el que el avión se encuentre en el suelo antes de despegar (Ground = True) esto se hace con el fin de asegurar que escogemos el valor correcto ya que hay algunos vuelos que les faltan algunos mensajes al despegue y aterrizaje.

- **Touchdown:** para la obtención de este punto se ha seguido un proceso similar al descrito para el Takeoff, salvo que los puntos a identificar son los relacionados al aterrizaje por tanto necesitaríamos el ultimo mensaje en el que el avion se encuentra en el aire (Ground = False) y el primer mensaje en el que el avión se encuentra en tierra (Ground = True), se hará el mismo proceso de comprobación ya que en el aterrizaje también a veces hay falta de mensajes.

Para entender mejor el concepto anteriormente explicado facilitamos un gráfico que ilustra los puntos necesarios para poder determinar estos valores.

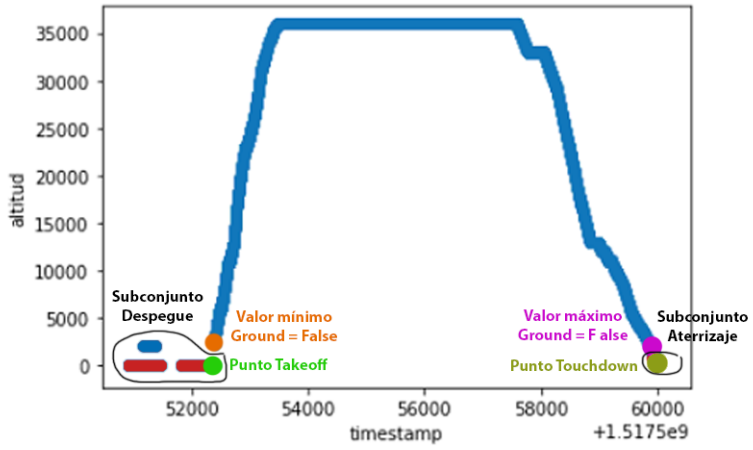


Figura 6.3: Gráfico Touchdown/Takeoff

Tras la obtención de estos valores, comenzamos barajando varios métodos para posteriormente realizar la predicción. En principio pensamos en alimentar al modelo con todas las características que teníamos de nuestros datos, y así obtener directamente el Touchdown como valor a predecir. El problema es que el timestamp es especifico de cada vuelo, por tanto nuestro modelo no generalizaba bien otros vuelos realizados en diferente momento. El modelo asocia las condiciones de velo a un timestamp concreto, pero no es capaz de extrapolar estas condiciones a un timestamp futuro asociado a otro vuelo diferente.

Por eso decidimos probar con un segundo enfoque que consistía en calcular el tiempo restante hasta el Touchdown en función del punto en el cual nos encontrábamos. A este valor lo llamaríamos RTA (*Required Time of Arrival*). El planteamiento que proponemos es basarnos en un valor artificial como es el RTA, en lugar del timestamp que es especifico de cada vuelo. Gracias al RTA, todos los vuelos presentarán unos valores similares a diferencia del timestamp el cual tenia valores muy dispares. Permitiendo así al modelo asociar una serie con las condiciones de vuelo que contengan un mismo valor de etiqueta.

Este planteamiento es más interesante porque podríamos plantear las predicciones con diferentes ventanas de tiempo, siempre teniendo en cuenta que estas ventanas dependen

del número de mensajes disponibles. Podremos realizar predicciones a dos horas vista, una hora vista, media hora y quince minutos. Lo que esperábamos con esto era observar que cuanto mas avanzaba nuestro vuelo en su ruta, y este generaba mas datos para nuestro modelo, las predicciones mejoraban.

Ahora ya contábamos con todos los datos que queríamos para poder seguir adelante con el estudio. Ahora necesitábamos hacer un estudio de los datos para poder seleccionar los atributos que tuvieran una correlación más fuerte entre ellos, por ello realizamos un estudio de correlaciones.

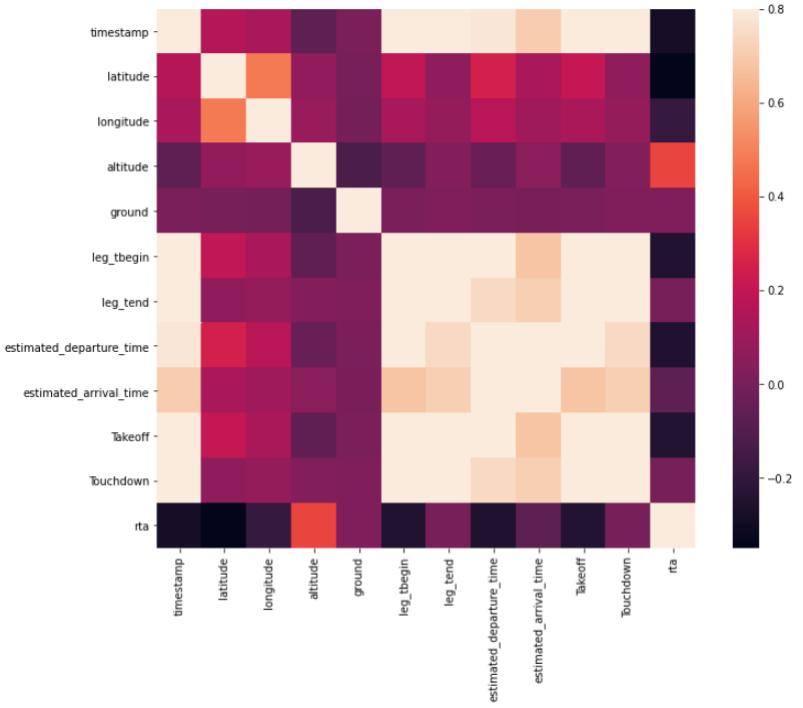


Figura 6.4: Matriz de correlaciones

Esta matriz representa la correlación entre las características, cuanto mas se acerca al color negro mayor es la correlación existente y cuanto mas se acerca al blanco menor es. Lo interesante es que según este estudio los datos que mayor correlación tenían con la variable RTA (y por tanto deberían ser más útiles para predecir su valor) eran los siguientes:

	rta
rta	1.000000
altitude	0.352526
ground	0.019697
leg_tend	-0.001885
Touchdown	-0.002827
estimated_arrival_time	-0.069740
longitude	-0.187451
Takeoff	-0.242761
leg_tbegin	-0.246984
estimated_departure_time	-0.253904
timestamp	-0.285731
latitude	-0.350025

Figura 6.5: Ranking correlaciones

Las variables con mayor correlación son altitud y ground. La altitud ya la habíamos considerado como características a usar. Gracias a este estudio añadiremos ground a las características consideradas. También consideramos que latitud, longitud son importantes a la hora de realizar la predicción, aunque no se encuentren en posiciones altas en el gráfico probablemente porque estos no varían mucho respecto a los demás valores, si consideramos que tienen importancia en el modelo. La tabla ayuda a considerar algunas características que no hayamos considerado, pero siempre tendrá prioridad nuestro criterio.

Antes de entrenar nuestra red neuronal con los datos, nos dimos cuenta de que los valores de la izquierda y de la derecha de la distribución que pertenecen al despegue y aterrizaje podrían influir a la hora de la predicción ya que son valores repetitivos en el cual aparece que el avión esta en la pista, así que, los suprimimos dejando solo el primer punto en el que el avión despega, que en nuestro caso es el Takeoff, y el ultimo punto al aterrizar que es el Touchdown.

Las redes neuronales tienen una particularidad y es que los datos necesitan ser normalizados, en caso de tener algún dato categórico (que no tenga valor numérico). Y una vez transformados se tienen que escalar. El escalado nos permite que todos los valores se encuentren en un mismo rango, en nuestro caso (0,1) con esto evitaremos que atributos con un dominio de valores muy grandes predominen sobre los atributos con valores muy pequeños.

Para esta primera tarea utilizaremos LabelEncoder que nos proporciona Python en sus librerías, con el que podremos realizar esa transformación de una variable categórica a numérica. Por ejemplo, en el caso de los identificadores de vuelos supongamos que tenemos el valor *RYR5NF_4CAA5A_1517571703_1517577271* este pasara a ser 1 el siguiente identificador de vuelo pasara a ser 2 y así consecutivamente.

El siguiente paso que vamos a realizar es dividir nuestros datos en 2 subconjuntos: uno servirá para entrenar la red neuronal y el otro servirá como conjunto de evaluación para que después de obtener el modelo entrenado, este pueda probarse con diferentes datos a los que se ha entrenado. La proporción que hemos decidido asignar ha sido un 80 % de los

datos para entrenamiento, y un 20 % para pruebas, se ha optado por esta distribución de los datos, por el volumen de vuelos con el que contamos.

En lugar de realizar el particionado de los datos de manera tradicional a nivel de registro/mensaje vamos a separarlas por series de mensajes de vuelo, así cada vuelo tendrá su conjunto de mensajes para entrenar y otro para evaluar.

Una vez divididos los datos podemos proceder a su escalado: para ello usaremos la función MinMaxScaler() que nos proporciona la librería de Tensorflow. Primero declararemos el escalador que nos permitirá llamar a la función *fit* y así determinaremos los mínimos y máximos, con el fin de aplicarlo posteriormente a los datos.

Una vez tenemos los datos distribuidos en dos subconjuntos y escalados necesitamos adaptarlos para nuestro problema, en este punto tuvimos la complicación por la cual tuvimos que extender el proyecto 1 Sprint más. A causa de que no sabíamos como hacer el siguiente paso, ya que había escasas documentación y la poca que había tenia un nivel técnico bastante alto por tanto no era fácil de entender y aplicarlo a nuestro caso. Uno de los principales retos que plantean las LSTM es el formateo correcto de los datos de entrada. En este sentido, el libro de Jason Brownlee [8] nos proporcionó una guía tanto para estructurarlos de la forma adecuada, como para adaptarlos a la implementación concreta de las LSTM en la librería utilizada, indicaba que necesitábamos tener 2 factores importantes uno era los puntos temporales que íbamos a retroceder hacia atrás a los que llamamos *timesteps* y los puntos en el futuro que íbamos a predecir que nosotros denominamos *predict_steps*. Por tanto, estos dos datos eran necesarios para poder reorganizar los datos de entrada que proporcionamos. Esto es posible gracias a la siguiente función:

```
def create_dataset(dataset, previous=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-previous-1):
        dataX.append(dataset[i:(i+previous), :len(features)])
        dataY.append(dataset[i + previous, len(features)])
    return np.array(dataX), np.array(dataY)
```

Esta función permite modificar nuestro dataset de manera que en vez de tener cada mensaje anotado con su RTA, tendremos secuencias de mensajes y estos a su vez anotados con sus secuencias de RTA's correspondientes, esto se debe a que las LSTM esperaran obtener los datos de esta manera. Estas secuencias se generan mediante una ventana deslizante, la cual deslizaremos 1 mensaje cada vez. De este modo cada secuencia de mensajes sera un registro en nuestro dataset. Para facilitar la comprensión, la (Figura 6.6) indica el proceso de la ventana deslizante, y de lo que hace la función. En esta se han tomado 50 mensajes hacia atrás para predecir uno hacia delante.

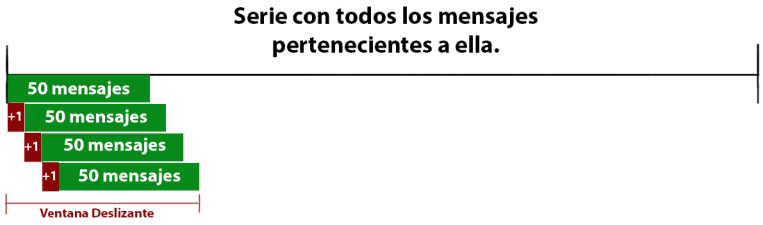


Figura 6.6: Ventana Deslizante

Una vez finalizado el proceso, los datos serán almacenados de manera matricial, por tanto tendremos una matriz de conjuntos de mensajes de esta manera:

```
(array([[[0.00323625, 0.68050102, 0.91153789, 1.        ],
         [0.01092233, 0.68032844, 0.91135307, 0.        ],
         [0.01436084, 0.68027815, 0.91129143, 0.        ],
         ...,
         [0.04753236, 0.67954055, 0.91142479, 0.        ],
         [0.05016181, 0.67953039, 0.91149368, 0.        ],
         [0.05238673, 0.6795201 , 0.91156614, 0.        ]],

        [[0.01092233, 0.68032844, 0.91135307, 0.        ],
         [0.01436084, 0.68027815, 0.91129143, 0.        ],
         [0.01618123, 0.68025461, 0.91126307, 0.        ],
         ....
```

Figura 6.7: Datos después de transformación

La función nos devolverá 2 conjuntos de datos, DataX y DataY. En el subconjunto X tenemos los datos de las características que hemos seleccionado en los cuales se va a basar para predecir el valor, mientras que en Y se encuentra el valor a predecir.Ahora que tenemos todos los factores necesarios se procede a preparar los dos conjuntos de datos Train y Test.

Comenzando con train lo primero que haremos será hacer una comprobación de la serie que vamos a preprocesar, esta debe tener los mensajes suficientes para poder realizar la ventana deslizante, si los tiene añadiremos la serie a *lista_X*

Para nuestro propósito, los datos de entrenamiento deben ser formateados acorde a la estructura definida. En este caso tenemos las etiquetas del conjunto de entrenamiento en forma de array bidimensional, sus dimensiones representan: Número de mensajes, Número de características. En cuanto a nuestro dataset original, seleccionamos las características elegidas para entrenar y convertimos los valores en numéricos, siempre que esto sea necesario y acto seguido los transformamos como hemos descrito antes.

Después aplicaremos la función descrita antes *create datraset*, esta nos proporcionara un array tridimensional, las dimensiones estarán formadas por: número de mensajes, nú-

mero de mensajes que tomaremos hacia atrás, número de mensajes que tomaremos hacia delante.

Es importante aplicar una serie de filtros para asegurarnos de que las series disponen des suficientes mensajes para que estas puedan ser acopladas a la estructuras mencionadas.

6.3. Ajuste LSTM

El ajuste de una red neuronal puede ser un trabajo complejo en el cual se deben tener muchos factores en cuenta, desde el enfoque del problema, número de capas necesarias, tipo de capas que necesitas hasta disposición de los datos. En gran medida los resultados exitosos dependen de un buen análisis y parte en realizar pruebas y observar el comportamiento. En nuestro caso comenzamos con un número de datos pequeño, ya que entrenar una red neuronal conlleva tiempo y cuantos más datos y características le añadas al problema más tarda en realizar la fase de entrenamiento. Con los datos de test, se seguirá el mismo proceso.

En análisis y predicción de series temporales con modelos LSTM es común comenzar con una red neuronal denominada *vanilla*. Es un modelo LSTM que tiene una sola capa oculta de unidades LSTM y una capa de salida utilizada para hacer una predicción. La clave de estas reside en la forma en la que los datos deben ser proporcionados, ya que el modelo necesita conocer el número de pasos hacia atrás y el número de pasos hacia delante que pretendemos predecir por cada conjunto de datos. Las LSTM *Vanilla* están compuestas por 50 unidades LSTM.

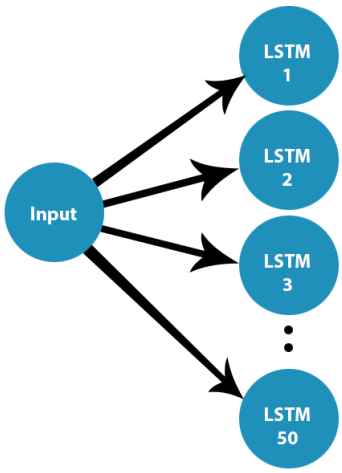


Figura 6.8: Estructura LSTM vanilla

Con este modelo, el problema que teníamos es que la curva de aprendizaje descendía muy rápido por tanto no se producía un aprendizaje óptimo. Esto se debe a que la tasa de

aprendizaje era muy grande. Después de estos resultados decidimos descender el número de neuronas de la primera capa de 50 a 10, con esto ganaríamos agilidad a la hora de entrenar el modelo, ya que cuantas más capas y neuronas más lento es el proceso de entrenamiento, y la otra razón es para intentar suavizar esa curva de aprendizaje.

También probamos varios optimizadores. Estos son los encargados de ajustar la función de pérdida para la retroalimentación de la red y así ajustar los pesos para que cada vez disminuya más la perdida de nuestro modelo por cada ejemplo. En el optimizador Adam[1], el cual usa el descenso del gradiente estocástico, lo que hace es encontrar mínimos y máximos por cada iteración. Tenemos que tener en cuenta cómo de dispersos están nuestros datos entre si: en nuestro caso los datos están muy cercanos unos al otro por tanto decidimos probar a disminuir la tasa de aprendizaje con el fin de controlar la magnitud de los cambios que realiza nuestro optimizador en el tiempo, al estar los datos juntos y la tasa más baja esto permitirá obtener mejores resultados.

A la hora de entrenar el modelo debemos tener en cuenta dos factores importantes denominados *lotes y épocas*. Los primeros corresponden a una estructura en bloque de vectores de entrada, estos son un subconjunto de la serie que vamos a entrenar,así controlaremos el número de muestras que la red verá antes de cada actualización de los pesos. Se debe tener en cuenta cómo de grandes va a ser estos "lotes" de datos, ya que si tenemos muchos datos para entrenar y entrenamos en lotes muy pequeños es posible que la red no aprenda con éxito o tarde mucho en ejecutarse. Como en nuestro subconjunto de entrenamiento tenemos 813.815 muestras probando diferentes configuraciones, hemos llegado a que el número de batch que mejor funciona para nuestra configuración es de 1000 unidades. En cuanto a las épocas, estas se refieren al número de iteraciones que vamos a realizar a la hora de entrenar nuestro modelo. Con estas iteraciones nos referimos a que el modelo use para entrenar todos los datos disponibles. En cuanto al número de iteraciones en nuestro caso tras realizar varias pruebas hemos fijado su valor en 25 ya que después de esa época el modelo ya no mejoraba su comportamiento.

Para evaluar cómo de bueno es un modelo usamos el Error cuadrático medio conocido como MSE. Consiste en medir la diferencia entre el valor real y el valor estimado, por tanto cuanto menor es el valor de esta mejor será el modelo ya que habrá menos diferencia entre los valores descritos.

6.4. Análisis de resultados

Tras entrenar nuestro modelo, se ha procedido a una serie de pruebas para verificar los resultados. En primera instancia hemos separado una serie que nuestro modelo nunca ha visto, por tanto, que no se ha usado para la fase de entrenamiento y hemos recortado la última hora de vuelo, con el fin de simular que el avión se encuentra en el aire y lo que le daremos a nuestro modelo como entrada es esa serie desde el despegue hasta ese último momento. Realizaremos esta prueba con diferentes franjas comenzando con 1 hora, 30 minutos y por último 15 minutos, lo que se espera es que cuanto menos tiempo

[1] https://keras.io/api/optimizers/adam/

quede para el aterrizaje y cuantos más datos tenemos del vuelo más acertadas serán las predicciones.

Lo que pretendemos realizar con esta prueba, es predecir cuánto estima el modelo que le queda a la aeronave para llegar al aeropuerto desde ese instante, por tanto, haremos una predicción a una hora vista. Para que el modelo pueda tomar los datos, estos deben ser sometidos a las mismas transformaciones descritas en el Capítulo 6.2 ya que si no el modelo no podría tratarlas. La serie que hemos seleccionado tiene el siguiente perfil de vuelo:

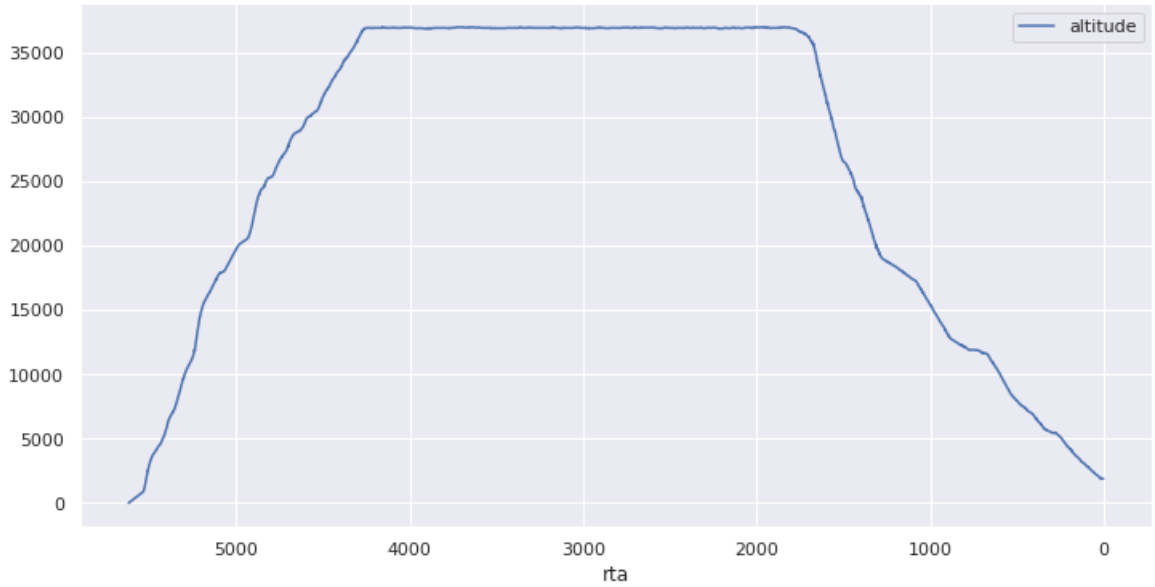


Figura 6.9: Perfil vuelo serie de prueba.

Sobre nuestros datos de entrada tendremos que aplicar las mismas transformaciones descritas en la etapa de preprocesamiento, una vez realizada la predicción deberemos des-escalar los valores obtenidos para que estos tengan su magnitud original.

Alimentaremos al modelo con las mismas características usadas para el entrenamiento quitando la columna de RTA la cual queremos predecir. Una vez hecho esto ejecutamos nuestro modelo y esperamos a que nos devuelva las predicciones, las cuales unificaremos con nuestra columna de *RTA* para poder comparar como de buena ha sido la predicción. Las predicciones obtenidas son las siguientes:

	Pred RTA	altitude	longitude	latitude	ground	rta
0	5454.372070	0	2.60106	48.99826	True	5618
1	5486.280273	900	2.56440	48.99630	False	5532
2	5483.255371	1125	2.55970	48.99610	False	5527
3	5482.770996	1475	2.55356	48.99592	False	5522
4	5480.445312	1525	2.55260	48.99590	False	5522
...	...	...	...	...	...	...
755	3678.355713	37050	0.64470	45.90460	False	3728
756	3676.198242	37050	0.64380	45.90360	False	3727
757	3674.149170	37050	0.63530	45.89390	False	3722
758	3672.007812	37050	0.63459	45.89310	False	3722
759	3670.065430	37050	0.62540	45.88260	False	3717

760 rows × 6 columns

Figura 6.10: Predicción 1 hora vista.

Para entender cómo de buenas son las predicciones, debemos ir a la última fila de nuestra tabla. Es este último registro el que nos interesa, como podemos ver el valor real subrayado en azul era de 3717, y nuestro modelo ha predicho con color rojo 3670. Esto nos deja con un margen de error en la predicción de 47 segundos. Puede parecer mucho, pero teniendo en cuenta que es a una hora vista es un resultado muy bueno. Para comprender mejor este resultado vamos a realizar una pequeña visualización:

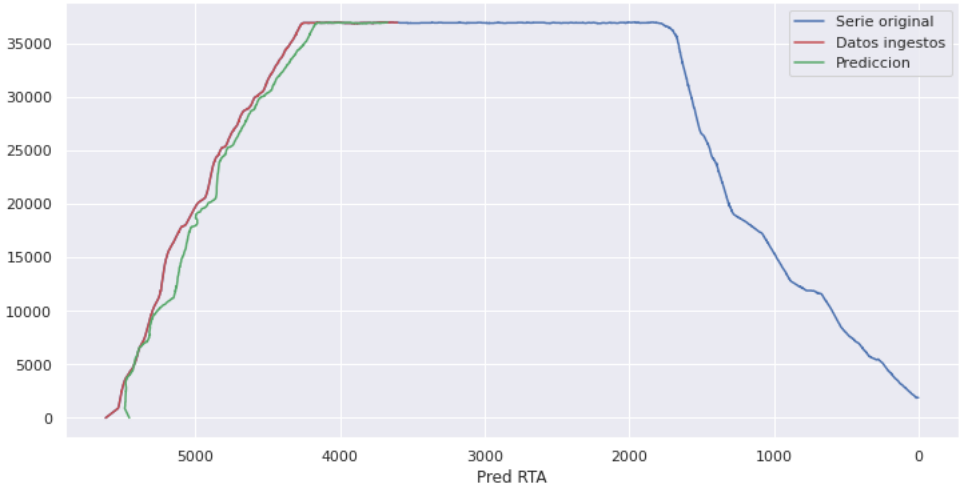


Figura 6.11: Gráfico predicción 1h

En este gráfico compuesto por eje X: RTA predicho y eje Y: altura, podemos diferenciar varias fragmentos, de color azul tenemos la serie original como era sin recortar. En rojo tenemos la serie una vez esta fue recortada, es esta la que se ha facilitado al modelo como entrada para realizar la predicción. Y en color verde tenemos la predicción realizada por nuestro modelo. Como podemos ver, al principio de la serie la predicción no es tan precisa, esto se debe probablemente a que el valor que hay que predecir es muy grande. Según avanzara en nuestra predicción ira mejorando, ya que cuanto mas nos acerquemos al aterrizaje menor es el valor de RTA y por tanto mas precisa nuestra predicción.

Para simplificar un poco la visualización y ver cómo de buena ha sido la predicción, vamos a ver un gráfico en el que pondremos en el eje X los RTA predichos y en el eje Y los RTA reales, cuanto más recta es la linea mejor es la predicción que se ha realizado.

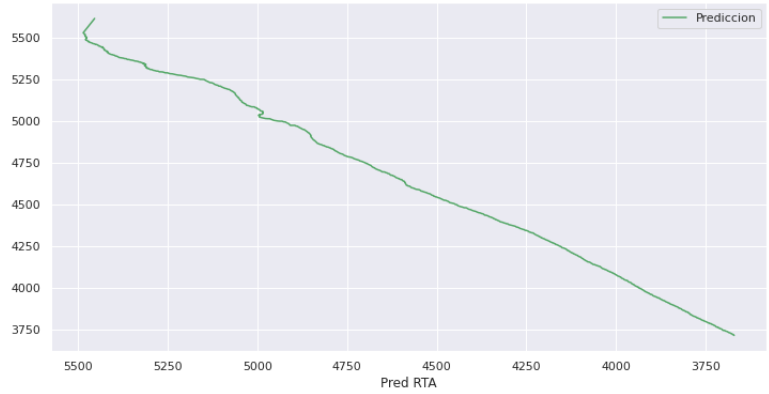


Figura 6.12: Rta real vs Rta Predicho

Vamos a realizar el mismo proceso ahora con el vuelo a 30 minutos vista, al ser un valor mas próximo al aterrizaje este debería de ofrecer una predicción mas precisa.

	Pred RTA	altitude	longitude	latitude	ground	rta
0	5454.372070	0	2.60106	48.99826	True	5618
1	5486.280273	900	2.56440	48.99630	False	5532
2	5483.255371	1125	2.55970	48.99610	False	5527
3	5482.770996	1475	2.55356	48.99592	False	5522
4	5480.445312	1525	2.55260	48.99590	False	5522
...	...	...	...	...	...	...
1519	2024.361938	37000	-1.97392	42.32785	False	1947
1520	2023.308716	37025	-1.97580	42.32500	False	1942
1521	2021.524658	37025	-1.98652	42.30886	False	1938
1522	2020.069702	37000	-1.98370	42.31320	False	1937
1523	2019.465576	37025	-1.99057	42.30276	False	1935

1524 rows × 6 columns

Figura 6.13: Predicción 30 minutos vista.

Después de ejecutar todo, los resultados obtenidos son: valor predicho 2019, valor real 1935. Esto hace un total de 84 segundos de desviación, a pesar de que el modelo tenía más datos dio una estimación algo peor que en el anterior resultado.

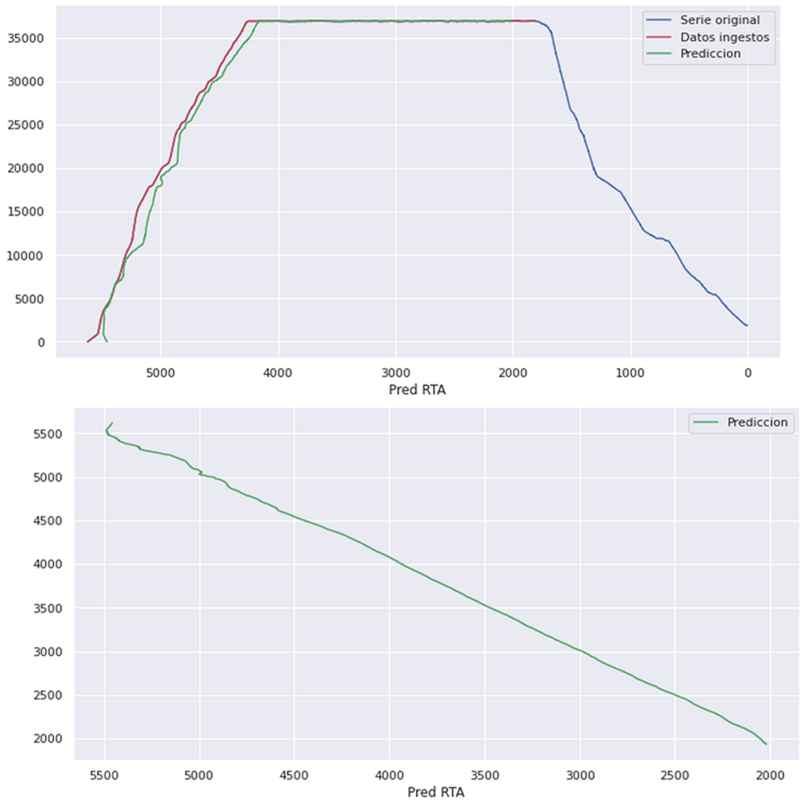


Figura 6.14: Gráficos predicción 30 minutos.

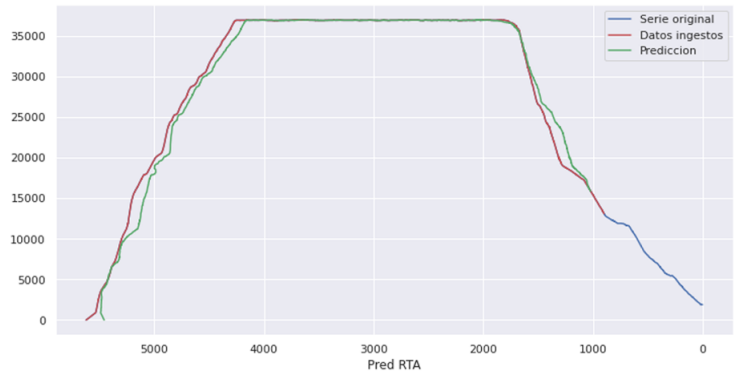
Por último concluimos este análisis con la franja de los últimos 15 minutos. Este nos ha dado el conjunto de datos de la Figura 6.15, en los cuales podemos apreciar que el ultimo valor real es de 1022 y el valor predicho por nuestro modelo es de 1024, por tanto difiera solo en 2 segundos. Es un resultado bueno teniendo en cuenta la poca diferencia que hay entre el valor real y el predicho.

	Pred RTA	altitude	longitude	latitude	ground	rta
0	5454.372070	0	2.60106	48.99826	True	5618
1	5486.280273	900	2.56440	48.99630	False	5532
2	5483.255371	1125	2.55970	48.99610	False	5527
3	5482.770996	1475	2.55356	48.99592	False	5522
4	5480.445312	1525	2.55260	48.99590	False	5522
...	...	...	...	...	...	...
1901	1032.531616	16175	-2.75010	40.54390	False	1032
1902	1031.318970	16100	-2.74719	40.54071	False	1030
1903	1026.673096	16050	-2.74480	40.53810	False	1027
1904	1025.371338	15975	-2.74157	40.53448	False	1024
1905	1024.350952	15900	-2.73940	40.53210	False	1022

1906 rows × 6 columns

Figura 6.15: Predicción 15 minutos vista.

Visualizando los gráficos podemos observar que se ajusta muy bien. La línea representada en el gráfico que muestra los valores de RTA reales vs valores de RTA predichos es muy recta, por lo tanto, podemos decir que hemos obtenido unos resultados satisfactorios:



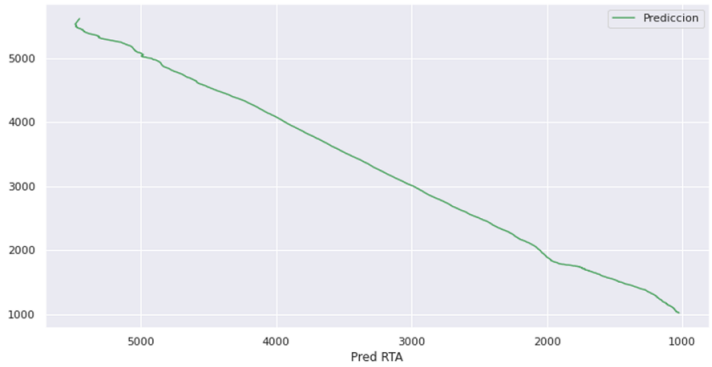


Figura 6.16: Gráficos predicción 15 minutos.

En general la precisión de los resultados puede variar según la serie, el número de mensajes que la componen, ya que hay series que tienen trayectos más pequeños por lo cual contamos con menos datos para poder realizar la predicción. La siguiente prueba que realizamos, fue hacer este mismo estudio masificado a todas las series del conjunto de test de manera automática, calculando el error absoluto medio y el error relativo medio. En total el conjunto de test consta de 104 vuelos los resultados obtenidos fueron los siguientes:

FRANJA	ERR.ABSOLUTO(SEG)	ERR.RELATIVO	RTA PRED MEDIO	RTA MEDIO
1H	671	0.10927294649039047	7934.63	8562.67
30 MIN	271	0.10177113714803569	6164.34	6416
15 MIN	146	0.09256106512725783	5267.92	5458.03

Figura 6.17: Tabla de resultados

Capítulo 7

Construcción del Dashboard

El siguiente capítulo, lo dedicaremos a explicar cómo se ha llevado a cabo la construcción de nuestra aplicación en forma de dashboard o panel de mandos. Con el fin de crear algo tangible que plasme el conocimiento que hemos adquirido durante la fase de estudio del proyecto y así aplicar estos conocimientos. Vamos a explicar cómo se ha desarrollado la aplicación y hablaremos de sus arquitecturas tanto lógica como física, también de como lanzar la aplicación a producción desplegándola en un servidor.

7.1. Análisis

En esta sección realizaremos un análisis sobre el proyecto. Vamos a desarrollar una aplicación web que nos permita visualizar esa gran cantidad de datos y poder así usar de manera sencilla nuestro modelo anteriormente desarrollado. Por esta razón, debemos realizar una especificación de las historias de usuario que definirán como será la aplicación, y así analizar los diferentes requisitos que pueden surgir a lo largo de esta. El principal artefacto en el cual vamos a enfocar el análisis son los requisitos de usuario que describen la interacción del usuario con la aplicación.

7.1.1. Actores

Antes de comenzar con la especificación de los usuarios, debemos analizar qué actores podrán interactuar con el sistema y de qué modo. Nuestro sistema cuenta con dos actores:

1. ActR-01: ***Usuario Común.***
    - **Descripción**: Todo aquel que acceda a la plataforma tendrá el status de usuario común, ya que no se necesitará un rol diferente ni registro para poder usar los servicios de esta.
2. ActR-02: ***MapBox API***
    - **Descripción**: Este actor externo interactúa con el sistema proporcionando los mapas a los gráficos.

7.1.2. Especificación de requisitos

En esta sección vamos a especificar y desarrollar las diferentes requisitos a través de las historias de usuario, estas deben cumplirse para que podamos finalizar el proyecto con éxito. Podemos definir como historia de usuario como los mecanismos necesarios para construir un entendimiento compartido del proyecto, a través de la colaboración entre todas las partes que intervienen en él [37] . Las historias de usuario tienen este formato:

1. Para quién. **COMO** usuario del sistema.
2. Qué quiere. **QUIERO** un dashboard que me permita consultar los vuelos disponibles.
3. Para qué lo quiere. **PARA** para poder visualizar información de todos los vuelos disponibles en el sistema.

Junto a estas preguntas, debemos tener en cuenta los criterios de aceptación que debe cumplir dicha historia, para que se considere acabada de manera exitosa. Estas deben ser precisas y cuantificables. Se han identificado las siguientes historias:

1. **HUSU-1**: Seleccionar diferentes tipos de vuelo disponibles.
   - **Descripción:** Como usuario común del sistema. Quiero que el dashboard me permita consultar todos los vuelos disponibles, y que también me permita escribir, si así quisiera el identificador del vuelo manualmente.Para poder visualizar los vuelos y seleccionarlos de manera más fácil.
   - **Criterios de aceptación:**
     a) Debe poder buscarse un vuelo tanto escribiendo su identificador manualmente.
     b) Los gráficos se deben actualizar simultáneamente una vez se seleccione un vuelo.
2. **HUSU-2**: Evaluar calidad de la predicción.
   - **Descripción:** Como usuario común del sistema. Quiero poder visualizar en forma de gráfico el valor real de RTA y el valor predicho. Para poder analizar la desviación existente entre ambos y así poder evaluar de forma cualitativa la calidad de la predicción.
   - **Criterios de aceptación:**
     a) El gráfico debe ser interactivo para poder realizar lecturas de los valores en un rango o momento concreto.
     b) La linea de la predicción debe ser lo suficientemente grande como para poder ser vista con facilidad, así como su color se distinga del resto de elementos.

c) Los datos mostrados deben corresponder al vuelo seleccionado por el componente desplegable, además en cada actualizarse de este componente el gráfico debe actualizarse en tiempo real.

3. **HUSU-3**: Visualización de ruta de vuelo y predicción.

   - **Descripción:** Como usuario común del sistema. Quiero poder visualizar en un mapa la ruta que ha realizado el avión y además poder ver el valor predicho de RTA en cada punto. Para poder analizar la ruta que ha realizado el vuelo y poder visualizar la predicción hecha.
   - **Criterios de aceptación:**
     a) El gráfico debe ser interactivo para poder moverse libremente por el mapa y que además se pueda aproximar o alejar dentro de este para evaluar mejor la ruta.
     b) Al seleccionar algún punto dentro del trayecto del vuelo debe visualizar el valor de rta predicho por el modelo.
     c) Los puntos deben ser lo suficientemente grandes para poder seleccionarse.
     d) El mapa debe ser lo suficientemente grande para visualizar y operar con el.

4. **HUSU-4**: Visualización información complementaria del vuelo.

   - **Descripción:** Como usuario común del sistema. Quiero poder visualizar información sobre el origen y destino del avión. Para así poder saber con facilidad de dónde viene y hacia dónde se dirige.
   - **Criterios de aceptación:**
     a) Los datos deben visualizarse con el tamaño suficiente para poder verse con facilidad.
     b) Los datos deben comunicarse con los demás componentes por lo tanto cuando se seleccione otro vuelo estos deben actualizarse en tiempo real.

5. **HUSU-5**: Cargar ruta de vuelo.

   - **Descripción:** Como usuario común del sistema. Quiero cargar la ruta de un vuelo. Para así poder analizar esta posteriormente.
   - **Criterios de aceptación:**
     a) Para la subida de datos, es necesario que el sistema permita seleccionar el archivo que queramos subir.
     b) Este componente debe interactuar con los demás, por tanto una vez cargados los datos este debe actualizar los demás componentes.
     c) El recuadro debe ser lo suficientemente amplio para usar el drag and drop con facilidad para evitar soltar el archivo fuera de la caja.

6. **HUSU-6**: Visualizar el tiempo estimado de llegada.

   - **Descripción:** Como usuario común del sistema. Quiero poder visualizar el tiempo estimado de llegada de un vuelo cuando cargo sus datos de vuelo. Para así poder ver la predicción del modelo.
   - **Criterios de aceptación:**
     a) Los datos deben poder visualizarse con facilidad. Sin necesitar realizar zoom.

7. **HUSU-7**: Visualizar rutas con mapa de calor.

   - **Descripción:** Como usuario común del sistema. Quiero poder visualizar todas las rutas con sus respectivos mapas de calor en función de la diferencia entre los valores reales y los valores predichos . Para así poder evaluar donde se originan mas retrasos de manera visual.
   - **Criterios de aceptación:**
     a) No debe tardar en cargarse mas de 10 segundos.
     b) El mapa tiene que tener un tamaño lo suficientemente grande para poder operar con facilidad.

7.2. Diseño

En esta sección detallaremos el diseño previo al desarrollo del dashboard. Detallaremos la arquitectura lógica y la física usadas para desarrollar con éxito el proyecto, así como el diseño de la interfaz de nuestra aplicación web.

7.2.1. Arquitectura Lógica

A continuación vamos a detallar los componentes lógicos de nuestro sistema y cómo interactúan y se relacionan entre sí. La arquitectura lógica del proyecto se corresponde con la aplicación web desarrollada bajo el lenguaje Python usando la librería Dash, que a su vez es ejecutada sobre un servidor Flask. La arquitectura presentada es bastante sencilla, esto deriva del alto nivel del lenguaje Python y a su facilidad a la hora de desarrollo de este tipo de aplicaciones. Nuestro proyecto se corresponde con la siguiente arquitectura lógica:

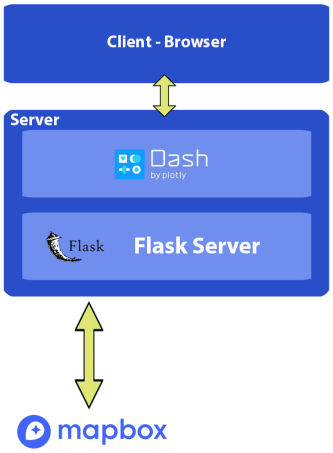


Figura 7.1: Arquitectura Lógica

En esta imagen podemos ver el servidor, en el tenemos una instancia de Flask sobre la cual se ejecuta Dash, a su vez esta interacciona con el navegador del cliente. Por otro lado nuestro servidor interacciona con una API externa que nos proporciona los mapas.

7.2.2. Arquitectura Física

En esta sección describiremos la arquitectura física usada a lo largo de este proyecto. Como nuestros datos cuentan con restricciones por la entidad que nos los proporciona *BRT-E* nuestra arquitectura física local es la siguiente:

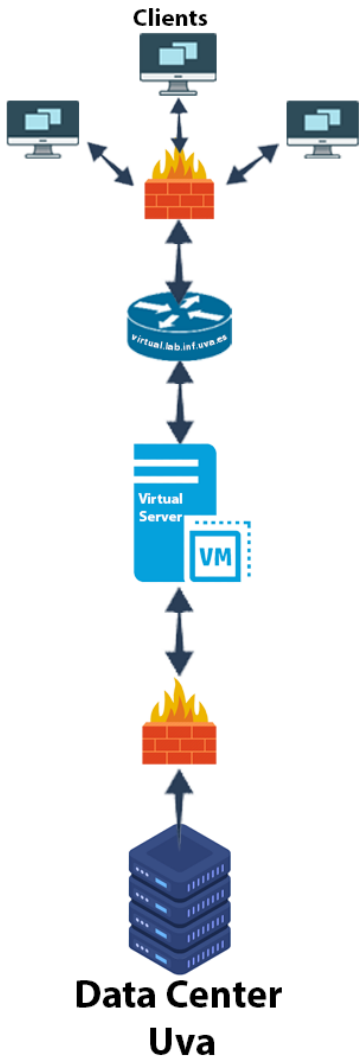


Figura 7.2: Arquitectura Física

En la Figura 7.2 podemos ver el CPD, sobre el tenemos instanciada una máquina virtual, en la cual tenemos desplegada la aplicación de manera local. Para interactuar con esta es necesario atravesar un cortafuegos y el Router el cual dirigirá las conexiones.

7.2.3. Diseño de Interfaz

En esta sección expondremos las interfaces con las que cuenta nuestra aplicación, como ya mencionamos estas van a ser sencillas. Hemos optado por pocas pestañas pero que aporten mucha información. La disposición principal de la aplicación se separa en tres apartados usando el componente *Tab* (pestaña) el cual nos permitirá movernos entre funciones.

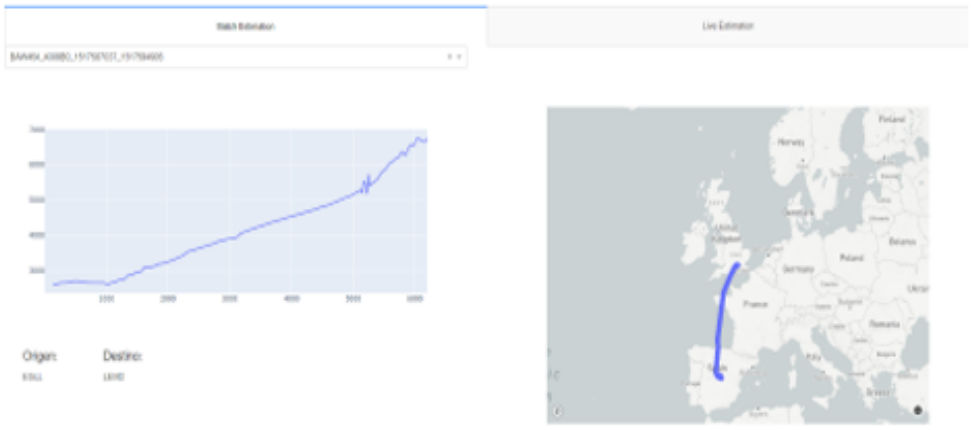
Titulo	Inicio/Batch Estimation
Descripción	Pantalla principal de la aplicación, en esta podremos visualizar los vuelos disponibles comprobar su ruta, origen-destino y la predicción en cada punto de la ruta del tiempo estimado.
Activación	Al inicio de la aplicación.
Diseño	

Figura 7.3: Interfaz 1 Inicio/Batch Estimation

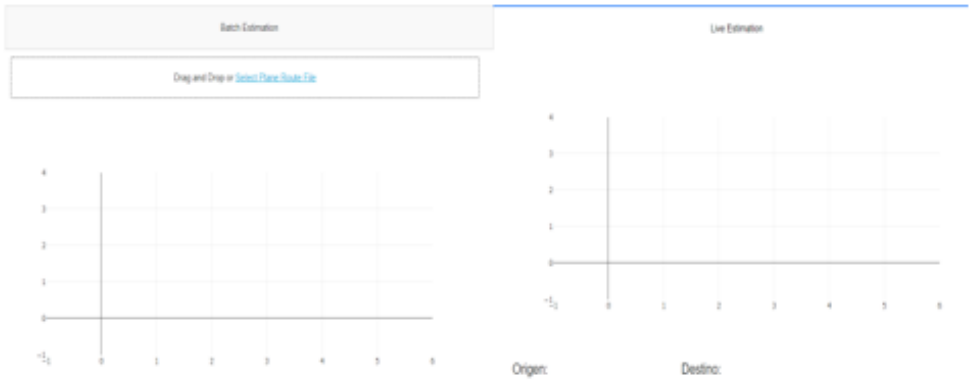
Titulo	Live Estimation
Descripción	Simula lo que visualizaríamos cuando el avión esta en vuelo, tanto a nivel de ruta como nivel de predicciones.
Activación	Cuando el usuario presione sobre la pestaña Live Estimation. Es necesario cargar el archivo .csv con la ruta actual del avión.
Diseño	

Figura 7.4: Interfaz 2 Live Estimation

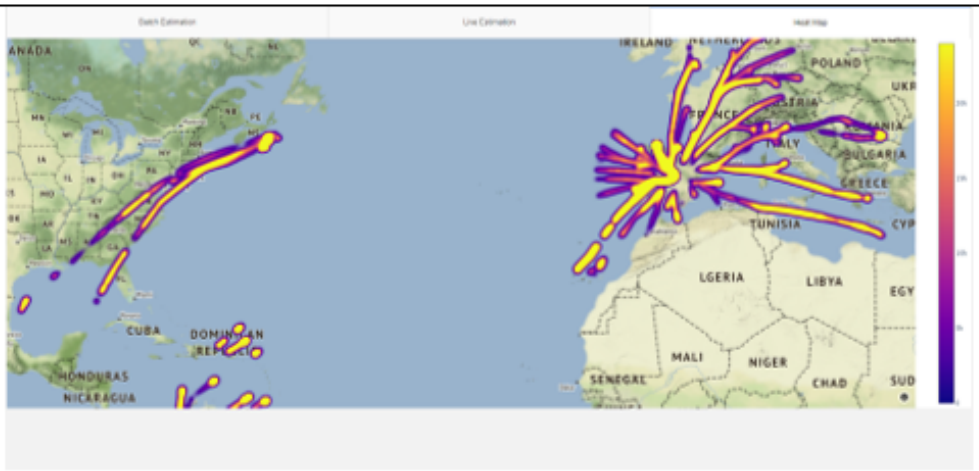
Titulo	Heat map
Descripción	Visualización de todas las rutas, estas aparecerán indicadas con colores en función de como de grande es la diferencia entre la predicción y el valor real.
Activación	Cuando se acciona la pestaña Heat map
Diseño	

Figura 7.5: Interfaz 3 Heat Map

7.3. Implementación

En esta sección se detallará en profundidad cómo ha sido implementado el sistema en este proyecto. Veremos cómo se ha desarrollado la aplicación web y cómo se ha desplegado posteriormente a un entorno de producción, así como todas las herramientas necesarias para el desarrollo de la aplicación.

Hemos elegido desarrollar la aplicación web en el lenguaje de programación Python porque es un lenguaje muy versátil. Además nuestro modelo de aprendizaje automático estaba desarrollado en Python y esto facilitaría mucho las cosas sobre todo a la hora de usar librerías e integrar el modelo ya que nos proporcionarían una base y así no tendríamos que construir desde cero los componentes individuales de la red. Aunque hemos usado Dash como framework para la construcción de la parte visual se necesitan conocimientos sobre maquetación web en concreto HTML,CSS y React. Por la parte del servidor debemos tener conocimientos en Flask, nuestra intención era desarrollar una aplicación de una sola página de ahí que se necesiten estos conocimientos ya que el flujo de información se producirá de manera asíncrona permitiendo que se produzca ese efecto de una sola página.

7.3.1. Entorno de desarrollo

Este proyecto ha sido desarrollado en dos fases. La fase perteneciente al estudio y modelización de la red neuronal, y la fase de creación y despliegue de la aplicación. Ambas se han se han desarrollado en un sistema perteneciente a la familia de Linux, Ubuntu 18.04 alojado en el *data center* de la Universidad de Valladolid, esta máquina dispone de un procesador de 8 núcleos y cuenta con 16GB de ram.

El lenguaje de desarrollo en ambas fases ha sido Python concretamente la versión 3.7, a pesar de que Python está instalado dentro de las dependencias de Ubuntu, la versión que está pre-instalada es más antigua , por tanto tuvimos que instalar esta última.

1. Instalación entorno de desarrollo

Con el fin de ahorrar algunos pasos hemos decidido usar el entorno de desarrollo proporcionado por Anaconda Python. Además esta distribución es muy usada en ciencia de datos, aprendizaje automático etc... Nos proporcionará Python en su versión 3.7 y además Jupyter Notebooks que es un entorno que nos facilita realizar estudios, gracias a su formato en forma de cuaderno y sus ejecuciones controladas. Ambas vienen integradas en un mismo instalador. Para la descarga de Anaconda en entornos Linux debemos hacer estos pasos:

- Descargar el programa de su repositorio, para ello usamos este comando: *curl –O https://repo.anaconda.com/archive/Anaconda3-2020.02-Linux-x86_64.sh*
- Ejecutamos el archivo de instalación: *bash Anaconda3-2020.02-Linux-x86_64.sh*
- Ejecutamos el programa: *anaconda-navigator*

Una vez realizado estos pasos nos abrirá una ventana desde la cual podremos ejecutar tanto Jupyter Notebooks como otros IDE como Spyder, por ejemplo. Una vez ejecutemos Jupyter y creamos nuestro cuaderno deberíamos ver lo siguiente:

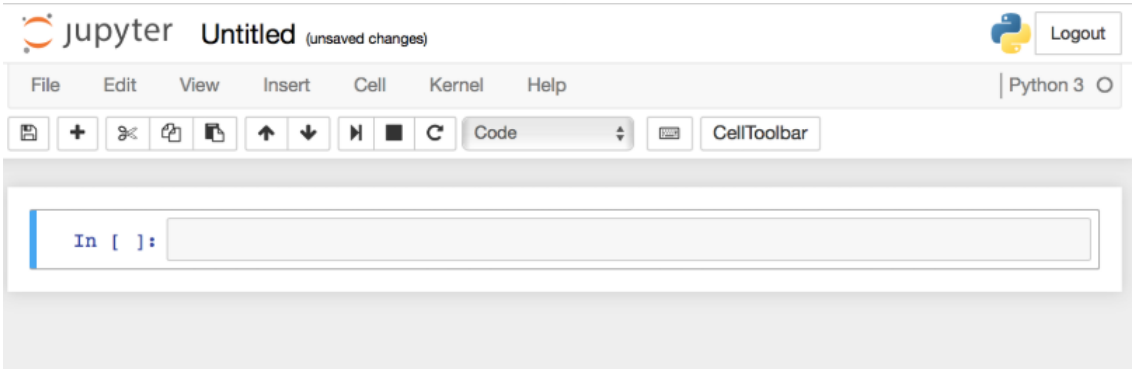


Figura 7.6: Jupyter Notebook

Para la fase de creación y despliegue de nuestra aplicación web usaremos el IDE PyCharm, para su instalación en ordenadores con Ubuntu solo tendremos que ir al market de aplicaciones de ubuntu y debemos instalar *Jetbeans Toolbox*, es un gestor que nos permite instalar de manera fácil esta herramienta. Una vez hecho esto solo tendremos que instalar el gestor y desde el descargar la aplicación.

Una vez tenemos descargados nuestros IDE's procedemos a la descarga de las librerías necesarias tanto para desarrollar nuestro modelo como para la creación de nuestra aplicación web.

2. Instalación librerías

Con el paquete que viene instalado en la distribución de Anaconda tenemos muchas librerías por defecto, pero Python es un lenguaje el cual dispone de muchas más. Para llevar a cabo nuestro estudio vamos a necesitar librerías que no vienen instaladas por defecto. Estas son las siguientes:

- Pandas: Esta librería proporciona estructuras de datos y operaciones para poder manipular tablas y series temporales.
- Numpy: Librería que agrega soporte para estructuras vectoriales, matriciales y agrega funciones matemáticas de alto nivel para operar con los vectores y matrices.
- Random: Librería que nos permite generar números aleatorios.
- Matplotlib: Biblioteca que permite la generación de gráficos a partir de datos contenidos en arrays o listas.
- Seaborn: Librería de alto nivel basada en Matplotlib que provee de herramienta de visualización.
- Sklearn: Biblioteca de software libre que incluye algoritmos de aprendizaje automático.
- Tensorflow: Biblioteca de código abierto destinada al desarrollo de modelos de aprendizaje automático.
- Datetime: Biblioteca que permite la transformación en diferentes formatos de horas y fechas.
- Scipy: Biblioteca de código abierto que proporciona herramientas y algoritmos matemáticos.

Para la aplicación web debemos instalar las siguientes librerías adicionales aparte de las mencionadas anteriormente:

- Dash: Librería que nos permite usar los componentes del framework Dash.
- Plotly: Biblioteca que proporciona herramientas de análisis y visualización de datos en linea.

3.Herramientas de desarrollo

En esta sección definiremos todas las herramientas que hemos usado para el desarrollo de este proyecto.

- Jupyter Notebooks: Entorno informático interactivo creado en 2014 por Fernando Perez. Este se basa en la web, permite la creación de documentos de Jupyter notebook que soporta los siguientes formatos: HTML, diapositiva de presentación, LaTeX, PDF, ReStructuredText, Markdown, Python.

- Google Colabs: Entorno gratuito de Jupyter Notebook creado por Google, no requiere de ningún tipo de configuración previa para funcionar.

- PyCharm: Entorno de desarrollo creado en 2010 por la empresa Jetbrains, usado con el lenguaje de programación Python.

- Git: Software de control de versiones creado en 2005 por Linus Torvalds. Este software nos permite mantener un control de nuestra aplicación con diferentes versiones.

- Photoshop: Programa de edición gráfica creado en 1990 por la empresa Adobe. Permite la edición y manipulación de fotografías.

- Overleaf: Editor online que permite realizar documentos mediante el lenguaje de maquetación LateX.

4.Tecnologías de desarrollo

En esta sección vamos a definir todas las tecnologías y lenguajes de programación usados a la hora de desarrollar el proyecto.

Python: Lenguaje de programación multiparadigma diseñado por Guido van Rossum, este es capaz de soportar desde programación orientada a objetos, programación imperativa y programación funcional. Este se caracteriza por ser un lenguaje dinámico, interpretado y multiplataforma. Actualmente administrado por la Python Software Foundation. Destaca en ser un idioma versátil y con una infinidad de librerías creadas tanto por la fundación como por la comunidad.

Dash: Framework destinado a la creación de aplicaciones web interactivas con forma de panel de mandos. Funciona con Python implementado con Flask, Plotly.js y React.js.

Flask: Framework minimalista de Python que permite la creación de aplicaciones web con muy pocas lineas de código.

React.js: Es una Biblioteca del lenguaje de programación Javascript que permite el diseño y creación de interfaces de usuario, se especializa en aplicaciones de una sola pagina. Dentro del conocido patrón MVC o MVVM, React corresponde al desarrollo de la capa de la Vista.

7.3.2. Desarrollo de Interfaz

Para el desarrollo de la interfaz hemos usado un esquema parecido al que plantea HTML. La diferenciación es que dentro del framework de Dash existen unas etiquetas personalizadas para este cometido, pero el funcionamiento es el mismo. Para definir la estructura de nuestro proyecto lo primero que debemos hacer es inicializar un objeto de la clase aplicación de Dash.

```
airports = dash.Dash(__name__)
```

El segundo paso que debemos seguir es agregar una capa a nuestra aplicación, es un contenedor donde se alojaran nuestros componentes. Se realiza de la siguiente manera:

```
airports.layout = html.Div([#Contenido app])
```

Como hemos comentado antes podemos apreciar la etiqueta Div que proviene del lenguaje HTML pero antes de ella debo decir que forma parte de html, para que Dash lo interprete como tal.

En cuanto a los componentes, los que hemos usado en este proyecto han sido los siguientes:

- Tab: Este componente sirve para añadir pestañas y poder moverte entre ellas, su estructura se declara así:

```
    dcc.Tabs(id='tabs-example', value='tab-1', children=[
    dcc.Tab(label='Pestaña 1', value='tab-1'),
    dcc.Tab(label='Pestaña 2', value='tab-2'),
])
```

- Dropdown: Este componente nos permite usar una vista desplegable y a la vez poder hacer búsquedas escritas si así lo deseamos, se declara de la siguiente manera:

```
    dcc.Dropdown(
    id='demo-dropdown',
    options=[
        {'label': 'New York City', 'value': 'NYC'},
        {'label': 'Montreal', 'value': 'MTL'},
        {'label': 'San Francisco', 'value': 'SF'}
    ],
    value='NYC' #Valor por defecto
)
```

- Graph: Este componente es el que nos permite crear gráficos y poder interactuar con ellos, la estructura de un gráfico es al siguiente:

```
        dcc.Graph(
        id='example-graph-2',
        figure=fig
    )
```

En cuanto a la disposición de los elementos y el tamaño de estos, se puede usar la distribución columnar de Bootstrap, solo tenemos que indicar que queremos usar su librería externa como archivo css. Una vez hecho eso solo debemos añadir a cada componente de los arriba citados esto:

```
className='six columns'
```

En el ejemplo visto anteriormente señalábamos que el componente tenia que ocupar 6 columnas, la disposición columnar de Bootstrap indica que hay un total de 12 columnas por cada fila.

La estructura externa es fácil de entender,pero un poco difícil a la hora de visualizar, ya que los componentes están insertados en listas y cuando usas varios componentes separados por distintos Divs se complica su visualización en código.

7.3.3. Desarrollo de back-end

En esta sección vamos a ver la funcionalidad de nuestra aplicación, con el fin de entender cómo interaccionan los componentes entre la parte visual y la parte funcional. El primer componente que debemos entender es el llamado Callback, estos sirven de puente entre la parte front y la parte back de nuestro programa, en ellos definiremos que es lo que mandamos como entrada (Input), de dónde proviene y qué es lo que recibiremos como salida (Output) y a dónde van. Vamos a analizar su estructura:

```
@airports.callback(
Output('id-componente1', 'a donde va eje:figura1'),
Input('id-componente2', 'de donde viene'))
```

Debemos tener en cuenta que si vamos a hacer múltiples llamada estas deben ir dentro de una lista:

```
@airports.callback([
Output('id-componente1', 'a donde va eje:figura1'),
Output('id-componente2', 'figure2')],
[Input('id-componente4', 'de donde viene'),
 Input('id-componente5', 'dropdown')])
```

Estas llamadas tienen que ir seguidos de sus respectivas funciones de actualización. Deben ir siempre en parejas de dos primero el callback y acto seguido su función update:

```
@airports.callback(
dash.dependencies.Output("destino", 'children'),
[dash.dependencies.Input("legs-dropdown", "value")]
)
def update_graph(valueDropdown):
    selected_value = valueDropdown
    df = df[df["leg"] == selected_value]

    return df["airport_destination"].unique()
```

Los tipos de Callbacks que existen son:

- Un solo Input - Un solo Output.
- Un solo Input - Múltiples Outputs.
- Múltiples Input - Un solo Output.
- Múltiples Input - Múltiples Output.

Lo más importante a tener en cuenta es, dentro de la función de actualización, si tenemos múltiples Outputs siempre dejarlo indicado en el retorno de la función. Y también tener en cuenta cuál nos conviene en cada momento, ya que cuanto más componentes procesamos a la vez en un mismo Callback más compleja se hace su función de update.

7.3.4. Despliegue de la aplicación

Como plataforma de despliegue hemos elegido Heroku, que es una plataforma que ofrece servicios en la nube propiedad de Salesforce. Esta permite el despliegue de aplicaciones ofreciendo servicios escalables y seguros. Los lenguajes que soporta son: Java, Node.js, Scala, Clojure y Python, aunque en su primera versión solo soportaba el lenguaje Ruby.

Lo primero que tendremos que hacer crear una cuenta en Heroku en su pagina oficial. Una vez realizado este paso debemos descargar Heroku CLI, esto nos permitirá manejar las apps mediante la terminal. Ahora debemos añadir esta linea de código justo debajo de donde declaramos la creación de nuestra aplicación en Dash,esto nos permitirá indicar que el servidor debe ejecutar esta app.:

```
#Declaracion de dash
airports = dash.Dash(__name__)
server = airports.server
```

Para el siguiente paso debemos haber instalado todas las librerías que nuestra aplicación utiliza. Y además de esto, instalar el servidor que se ejecutará en el entorno creado en Heroku, para instalar el servidor usamos este comando:

```
pip install gunicorn
```

Ahora debemos crear dentro de nuestro directorio raíz un archivo .gitignore, este dirá a Git qué directorios deseamos ignorar, en nuestro caso son estos:

```
venv
*.pyc
.env
.DS_Store
```

El siguiente paso sera crear un archivo dentro del directorio raíz llamado Procfile: este permitirá a Heroku iniciar nuestra aplicación.

```
web: gunicorn airplanes:server
```

Una vez realizado el paso anterior, vamos a la terminal y nos situamos en el directorio de nuestro proyecto. Vamos a crear un documento requirements.txt el cual hará una lista de las librerías necesarias para ejecutar nuestra aplicación y sus versiones.

```
pip freeze > requirements.txt
```

Generado el archivo desde la terminal, accedemos en nuestra cuenta de Heroku mediante:

```
heroku login
```

El siguiente paso es opcional en caso de haber inicializado nuestro directorio en git, en caso contrario:

```
git init
```

Ahora vinculamos el directorio con el de Heroku:

```
heroku git:remote -a airports
```

Añadimos el contenido nuevo listo para ser subido al servidor y comprometemos los datos:

```
git add .
git commit -am "Inicio proyecto"
```

Lo último que debemos realizar es subir al servidor la versión mediante un push a la rama que deseemos en este caso la rama master:

```
git push heroku master
```

Este último proceso tardara unos minutos, ya que tiene que instalar todas las dependencias en el servidor y desplegarlas. Cabe destacar que en su versión gratuita se pueden hacer despliegues de hasta 500mb y 5 aplicaciones. Como este ultimo paso no lo hemos realizado por las limitaciones del proyecto no podemos aportar información gráfica del resultado final. Las ventajas que tiene esta plataforma es que nos permite hacer nuestra aplicación escalable y que esta se adapte en función de la demanda de sus usuarios, además de contar con un alto grado de seguridad, ya que Heroku usa los servidores de AWS. Como desventajas tiene que todos los servicios conllevan un coste adicional.

Capítulo 8

Manual de usuario

En este capítulo se desarrollará el manual de usuario de nuestra aplicación. Vamos a explicar el funcionamiento de nuestra aplicación, con el fin de facilitar el uso a nuestros usuarios. Nuestra aplicación es accesible desde cualquier navegador web, no se necesita instalaciones previas. Como esta aplicación tiene un carácter privado, ya que no se puede desplegar en internet, hemos obviado la parte de acceso controlado y registro a la aplicación. Nuestra aplicación dispone de los siguientes módulos:

Batch Estimation A la hora de acceder a nuestra aplicación nos encontraremos en esta pestaña ver Figura 8.1. En ella podremos ver un listado de todos los vuelos que disponemos en forma de desplegable ver Figura 8.2. Con dicho desplegable podremos seleccionar el vuelo que queramos visualizar o directamente escribir el identificador a mano, la aplicación filtrara los vuelos según vayamos completando el id. Una vez seleccionado el vuelo que queremos visualizar, nuestros gráficos se actualizarán. En el lado izquierdo tendremos un gráfico que nos indicará la calidad de la predicción mostrando así los valores reales en el eje Y, los valores predichos de RTA en el eje X, con el fin de poder hacernos una mejor idea de como de buena es la predicción. En ese mismo lado contamos con un panel que nos indicará el origen y destino del vuelo seleccionado. En la parte derecha encontraremos un mapa en el cual podremos visualizar la ruta, y sobre cada punto de esta el valor estimado de RTA generado por nuestro modelo. Debajo del mapa tendremos una barra en la cual podremos indicar cuanto queremos visualizar del trayecto ver Figura 8.3.

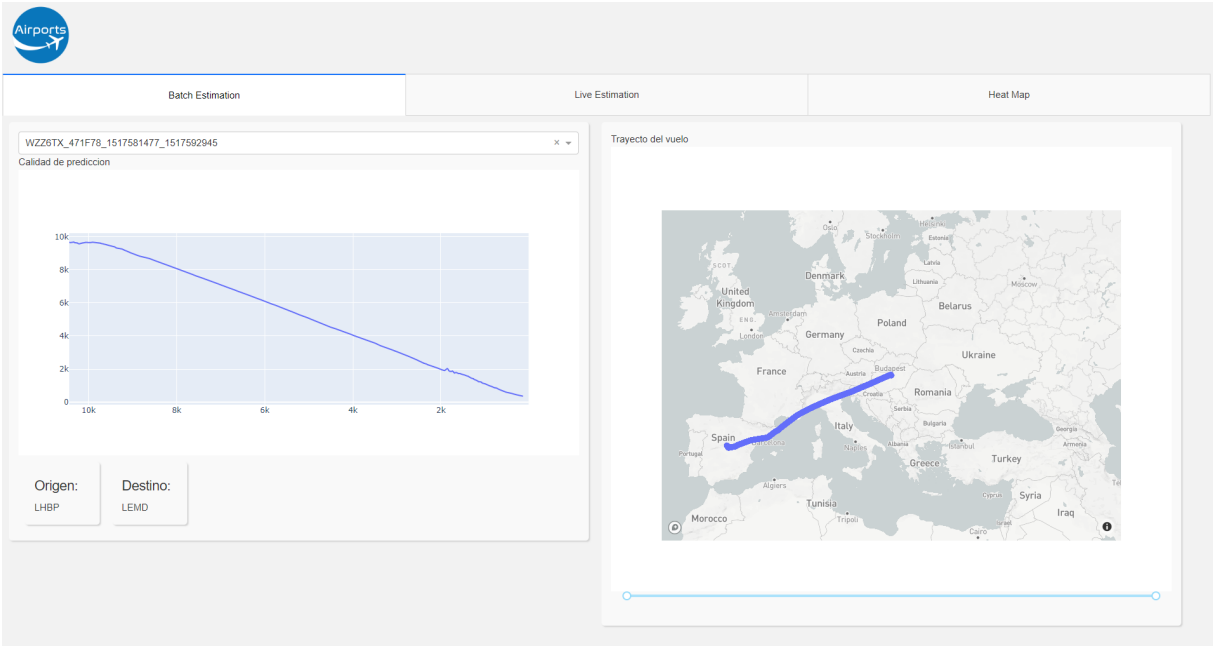


Figura 8.1: Pestaña Batch Estimation

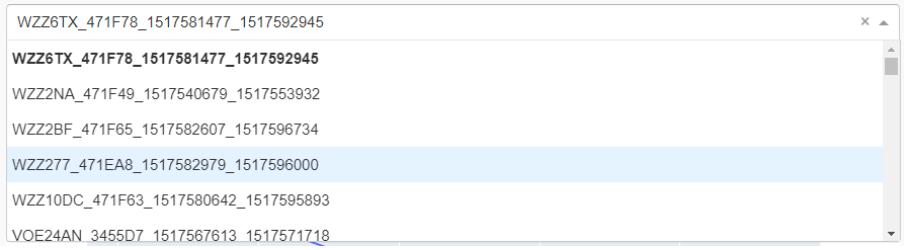


Figura 8.2: Barra desplegable

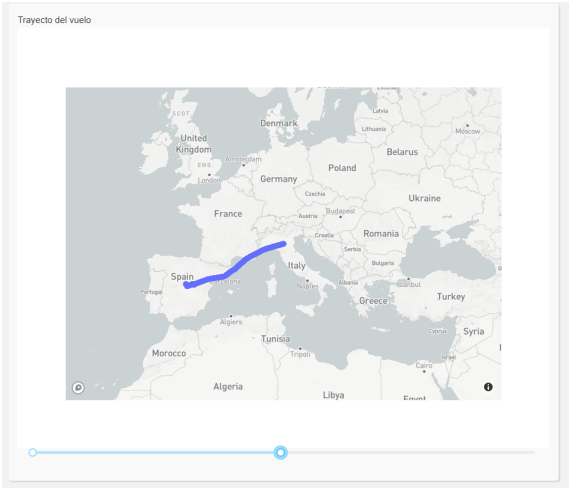


Figura 8.3: Trayecto y acotación

Live Estimation En la siguiente pestaña tenemos lo que seria la simulación de la estimación en tiempo real. Como no disponemos de una fuente que nos proporcione datos en tiempo real hemos simulado mediante la carga de un archivo csv. Este contiene los datos que nos proporcionaría una fuente de este tipo. En este apartado contamos con todo lo que teníamos en la pestaña de batch estimation, pero no tendremos la barra para elegir el fragmento del trayecto ya que se supone que estamos viendo en vivo la predicción. Los componentes nuevos son el campo para cargar archivos y el bloque que nos indica cuanto tiempo queda para llegar al aeropuerto destino (ver Figura 8.4).

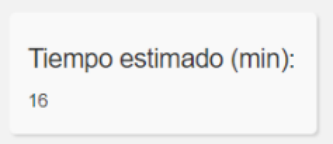


Figura 8.4: Tiempo estimado de llegada

Heat Map En esta pestaña podremos visualizar un mapa de calor, que nos mostrará la ruta de todos los vuelos y los retrasos sufridos en cada punto con un indicador de colores. Cuanto mas amarillo, peor ha sido la predicción en ese punto. Cuanto mas morado oscuro, mejor es la predicción en ese punto. Al ser tantos puntos si lo vemos muy alejado parece que el amarillo predomina por eso debemos hacer zoom (ver Figura 8.6) para que los puntos se separen un poco y que se pueda apreciar bien los colores de las predicciones.

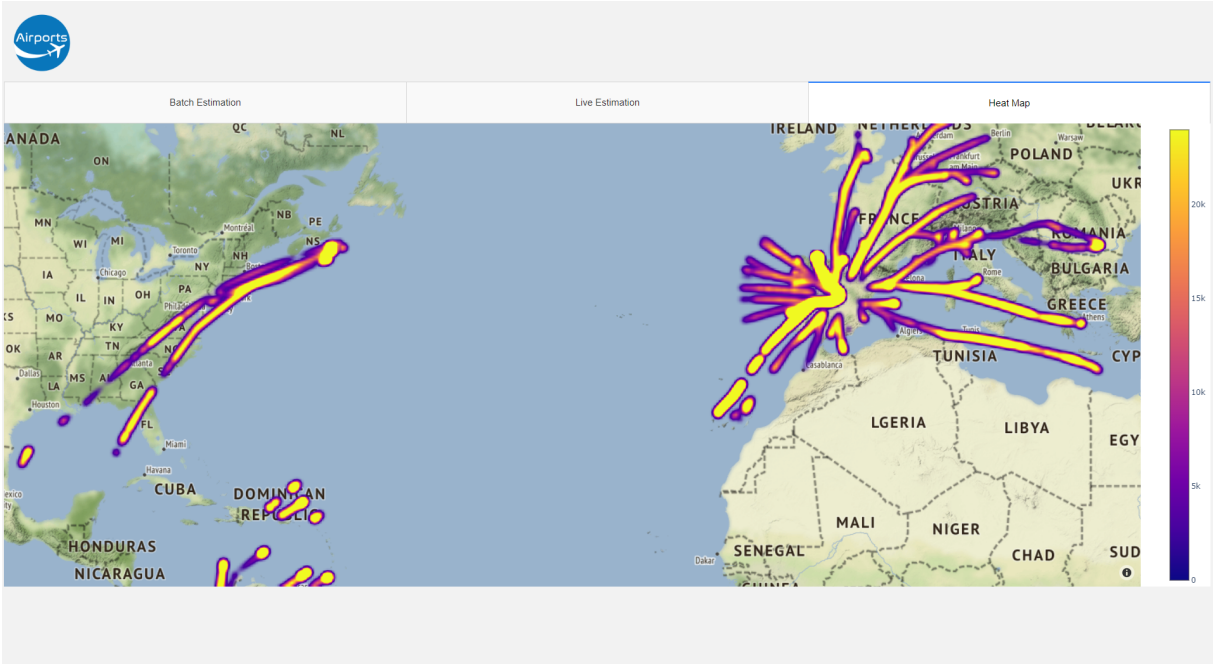


Figura 8.5: Mapa de calor

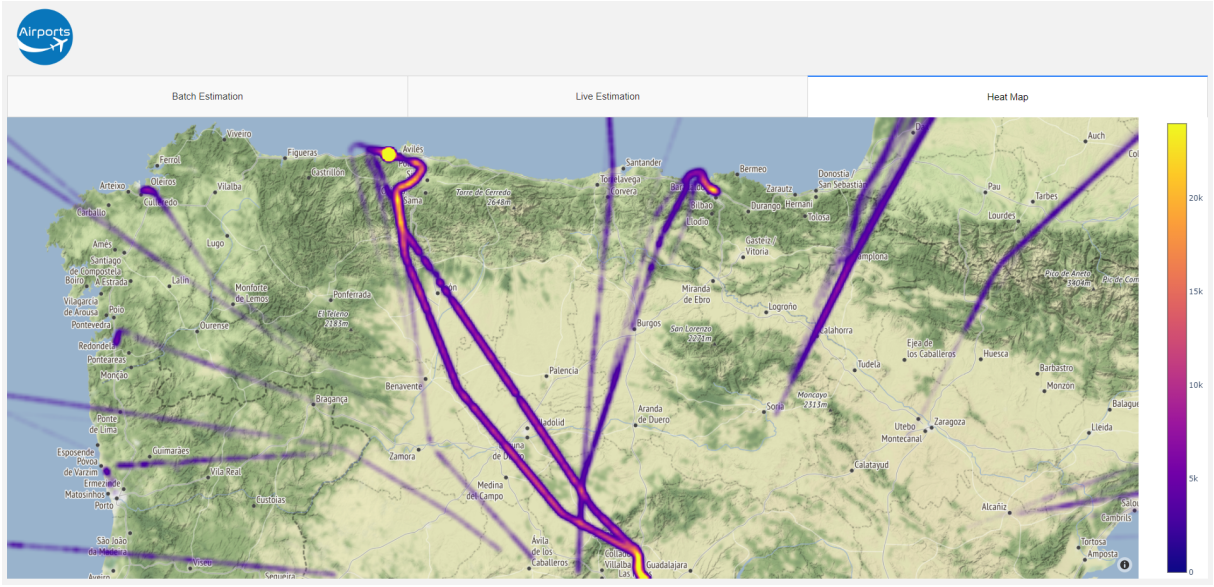


Figura 8.6: Mapa de calor con Zoom

Capítulo 9

Conclusiones y trabajo futuro

En este último capítulo presentaremos las conclusiones que hemos alcanzado tras el desarrollo de este proyecto. También se planteará una serie de posibles mejoras y ampliaciones que pueden derivar de este proyecto.

A lo largo del desarrollo de este proyecto me he encontrado con varias dificultades, entre ellas la más grande es el desconocimiento de las redes neuronales LSTM aplicadas a problemas de series temporales. Tener que lidiar con esta dificultad, me hizo investigar mucho sobre el funcionamiento de estas y su posterior implementación.

El segundo problema que encontré, que sin duda es uno de los puntos más importantes y con los que más aprendí, fue la adecuación de los datos a nuestro problema. Los datos tienen que ser tratados de una manera especial, con el fin de obtener el resultado deseado. Podríamos decir que sabía lo que quería obtener como resultado, pero no conocía las herramientas para llegar a ese resultado. Con esto último también tuve problemas ya que es un campo que todavía se encuentra en investigación y desarrollo, y por tanto no hay tantas fuentes de información, y mucho más aplicadas a la temática de este proyecto.

A lo largo del proyecto he aprendido mucho sobre la gestión de tráfico aérea y cómo esta funciona, sobre los problemas que se plantean derivadas de esa gestión. También he aprendido a trabajar con grandes volúmenes de datos y sobretodo a tratamiento de estructuras de datos ya que a la hora del pre-procesamiento de estos necesitaban muchas transformaciones.

Otra cosa muy importante que he aprendido a lo largo de este proyecto ha sido la visualización de la información y entender bien lo que me muestran los gráficos, creo que es algo fundamental ya que estos son los que dan valor a los resultados obtenidos.

Aunque en una medida menor he aprendido cosas que desconocía del lenguaje Python y sus diversas librerías. Por ejemplo, la creación de una aplicación capaz de visualizar información de manera tan intuitiva y cómoda y sin relativamente mucho esfuerzo gracias a Dash, del cual no tenía ningún conocimiento. Otra cosa asociada a esto último que he comentado, es a desplegar una aplicación en un entorno de producción en la nube, aunque no lo pude llevar a cabo del todo por las limitaciones que teníamos sobre el proyecto, pero sí pude hacer un despliegue sin subir los datos con datos ficticios.

9.1. Conclusiones

La conclusión principal que puedo sacar después de haber desarrollado este proyecto es que sin duda la inteligencia artificial es un campo puntero que ha llegado para quedarse, a lo largo de los próximos años llegará a ser un sector muy importante en nuestra sociedad. También puedo decir que creo que tecnologías como esta sin duda son capaces de hacer avanzar a nuestra sociedad, siempre sin olvidarnos de la importancia ética que conlleva. Este proyecto ha supuesto para mí un gran reto personal, a pesar de ser un tema que me gusta y el hecho de desear trabajar con este tipo de tecnologías tan punteras. El conocimiento obtenido y el aprendizaje constante al que me ha sometido este proyecto, sin duda ha sumado valor en mi perfil laboral ya que es algo que esta ahora mismo muy presente en el mundo laboral, tanto el Big Data como la Inteligencia Artificial.

Otra reflexión que extraigo del desarrollo de este proyecto es la gran complejidad que tiene nuestro trabajo, el gran número de variables a las que te enfrentas día a día y la complejidad sobre la que esta construido el mundo digital en general. Sobre la complejidad de adaptarse a una tecnología que está en constante evolución y sobre la investigación en este área.

A nivel de negocio la conclusión que puedo extraer es que, el sector aeronáutico a pesar de ser muy importante y que mueve una gran cantidad de capital, los costes asociados a problemas que provienen de la gestión del trafico aéreo, suponen un desembolso grande de dinero cada año, y hay pocas empresas que se dediquen a dar solución a las problemáticas que surgen. Las empresas existentes son empresas propias del sector. Y casi todos los estudios existentes son privados ya que son promovidos por este tipo de empresas.

A nivel técnico, no hemos mejorado las actuales modelos de predicción. Esto es debido por un lado a que estábamos condicionados por el alcance ya que el Trabajo Fin de Grado consta de 300 horas. Por otro lado, los datos de los cuales disponíamos eran limitados a un solo día y no contábamos con datos meteorológicos, los cuales habrían añadido un grado de dificultad aun mas alto pero sin duda habrían hecho que el modelo fuera más preciso. Con mas medios y mas tiempo probablemente se podría haber alcanzado un nivel similar al que tienen los modelos mas precisos.

9.2. Trabajo futuro

A pesar de que mi proyecto esté finalizado, existen una serie de mejoras que han surgido a lo largo del desarrollo y que no se han podido incluir porque nuestro alcance se limitaba por las horas de trabajo de un trabajo de fin de grado.

1. **Incorporar datos meteorológicos al modelo predictivo**: Como tenemos los datos de posicionamiento del avión, haría falta buscar una API externa de meteorología la cual nos diera información relativa al viento, presión atmosférica, precipitaciones etc... a lo largo de todos los puntos de nuestro trayecto. Esto sin duda mejoraría significativamente la predicciones.

2. **Predecir el estado del vuelo**: No solo predecir el tiempo restante para que el vuelo alcance el aeropuerto, sino poder predecir otros parámetros del vuelo como velocidad, altitud etc....

3. **Añadir análisis de los datos en tiempo real**: Obtener datos de un vuelo en tiempo real para que nuestro modelo realice la predicción del tiempo de llegada y podamos visualizar esta en tiempo real.

Bibliografía

[1] *Abandoning ACID in Favor of BASE in Database Engineering.* https://www.lifewire.com/abandoning-acid-in-favor-of-base-1019674. (Visitado 08-10-2019).

[2] *ACID or BASE databases? Consistency versus Scalability.* https://imelgrat.me/database/acid-base-databases-models/. (Visitado 08-10-2019).

[3] *Agile Manifesto.* http://users.jyu.fi/~mieijala/kandimateriaali/Agile-Manifesto.pdf. (Visitado 09-10-2019).

[4] *Análisis de series temporales: Modelos ARIMA.* https://addi.ehu.es/bitstream/handle/10810/12492/04-09gon.pdf. (Visitado 09-10-2019).

[5] *Apache Druid (part 1): A Scalable Timeseries OLAP Database System.* https://anskarl.github.io/post/2019/druid-part-1/. (Visitado 28-04-2019).

[6] *Box-Jenkins method.* https://en.wikipedia.org/wiki/Box%E2%80%93Jenkins_method. (Visitado 09-10-2019).

[7] Anibal Bregón. «Redes Neuronales». En: *Universidad de Valladolid* (2019).

[8] Jason Brownlee. *Deep Learning for Time Series Forecasting. Predict the Future with MLPs, CNNs and LSTMs in Python.* Autoria Propia, 2018.

[9] R. Calvo-Palomino y col. «Nanosecond-Precision Time-of-Arrival Estimation for Aircraft Signals with Low-Cost SDR Receivers». En: *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 2018, págs. 272-277.

[10] *Chaos Reprot.* https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf. (Visitado 10-11-2019).

[11] *Cielo Único Europeo.* https://www.enaire.es/sobre_enaire/presencia_internacional/cielo_unico_europeo. (Visitado 24-10-2019).

[12] *Database of databases.* https://dbdb.io. (Visitado 08-10-2019).

[13] *DB-Engines.*
https://https://db-engines.com. (Visitado 09-10-2019).

[14] *Dictionary coder.*
https://en.wikipedia.org/wiki/Dictionary_coder. (Visitado 09-03-2019).

[15] *Estos serían los trabajos más estresantes, según experta en selección.*
https://mba.americaeconomia.com/articulos/notas/estos-serian-los-trabajos-mas-estresantes-segun-experta-en-seleccion. (Visitado 28-10-2019).

[16] Noriko Etani. «Development of a predictive model for on-time arrival flight of airliner by discovering correlation between flight and weather data.» En: *Journal of big data* 1 (2019).

[17] *Extra Tree Classifier.*
https://towardsdatascience.com/an-intuitive-explanation-of-random-forest-and-extra-trees-classifiers-8507ac21d54b. (Visitado 07-11-2019).

[18] *Flightradar24.*
https://www.flightradar24.com/.

[19] Iván Garcıa y col. «Towards a scalable architecture for flight data management». En: *6th International Conference on Data Science, Technology and Applications.* 2017.

[20] *Gestión del trafico aéreo.*
https://greatbustardsflight.blogspot.com/2018/06/gestion-del-trafico-aereo-atm-de-forma.html. (Visitado 20-04-2019).

[21] *Graphite to monitor distributed systems.*
http://isselguberna.com/graphite-to-monitor-distributed-systems/. (Visitado 08-10-2019).

[22] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and Tensorflow. Concepts,Tools, and Techniques to Build Intelligent Systems.* O'REILLY, 2019.

[23] Sepp Hochreiter y Jürgen Schmidhuber. «Long Short-term Memory». En: *Neural computation* 9 (dic. de 1997), págs. 1735-80. DOI: 10.1162/neco.1997.9.8.1735.

[24] *How Sharding Works.*
https://link.medium.com/zjFNajBPA7.. (Visitado 28-04-2019).

[25] *Inefficient Airspace Cost the EU 17.6 bn in 2018.*
https://a4e.eu/europes-inefficient-airspace-cost-the-eu-e17-6-bn-in-2018-334-m-passengers-affected1/. (Visitado 24-10-2019).

[26] Ken Schwaber y Jeff Sutherland. *La guía de Scrum. La Guía Definitiva de Scrum:Las Reglas del Juego.* Spanish European, 2017.

[27] Adam Gibson Josh Patterson. *Deep Learning. A Practitioner's Approach.* O'REILLY, 2017.

[28] Claire J. Tomlin Kaushik Roy Benjamin Levy. «Target tracking and Estimated Time of Arrival (ETA) Prediction for Arrival Aircraft.» En: *AIAA Guidance* 1 (2006).

[29] *Key Concepts and Features of Time Series Databases.* https://www.alibabacloud.com/blog/key-concepts-and-features-of-time-series-databases_594734. (Visitado 18-11-2019).

[30] *Los retrasos aéreos en Europa aumentaron un 53 % en 2018.* https://www.preferente.com/noticias-de-transportes/noticias-de-aerolineas/los-retrasos-aereos-en-europa-aumentaron-un-53-en-2018-284093.html. (Visitado 24-10-2019).

[31] *More than 210.000 flights in Europe where delayed in June according to Eurocontrol.* https://www.airlines.iata.org/news/delays-in-europe-unacceptable-says-schvartzman. (Visitado 28-10-2019).

[32] Alexey Natekin y Alois Knoll. «Gradient Boosting Machines, A Tutorial». En: *Frontiers in neurorobotics* 7 (dic. de 2013), pág. 21. DOI: 10.3389/fnbot.2013.00021.

[33] Navaneeth Jamadagni Nathalie Kuhn. «Application of Machine Learning Algorithms to Predict Flight Arrival Delays». En: *University of Stanford* 1 (2017).

[34] Patrick O'Neil y col. «The log-structured merge-tree (LSM-tree)». En: *Acta Informatica* 33.4 (1996), págs. 351-385.

[35] *Run-length encoding.* https://es.wikipedia.org/wiki/Run-length_encoding. (Visitado 09-03-2019).

[36] Hanan Samet Samet Ayhan Pablo Costas. «Predicting Estimated Time of Arrival for Commercial Flights». En: *University of Maryland* 1 (2018).

[37] *SCRUM: Cómo escribir historias de usuarios sin morir en el intento.* www.tenstep.ec/portal/articulos-boletin-tenstep/41-scrum/253-scrum-como-escribir-historias-de-usuarios-sin-morir-en-el-intento. (Visitado 25-03-2020).

[38] *Time-series compression algorithms, explained.* https://blog.timescale.com/blog/time-series-compression-algorithms-explained/. (Visitado 28-04-2019).

[39] *Top 25 aeropuertos con más tráfico.* https://www.panynj.gov/content/dam/airports/statistics/statistics-general-info/annual-atr/ATR2019.pdf. (Visitado 24-10-2019).

[40] *Top time series databases.* https://www.influxdata.com/time-series-database/. (Visitado 08-10-2019).

[41] *Toyota Production System.* https://www.infoq.com/articles/scrum-sistema-producao-toyota/. (Visitado 07-11-2019).

[42] *Índice invertido.* https://es.wikipedia.org/wiki/%C3%8Dndice_invertido. (Visitado 25-03-2019).

Siglas

ACID Atomicity, Consistency,Isolation, Durability. 49, 50, 52, 58

ASM Aviation Services Management. 7

ATC Air Traffic Control. 7–12

ATFCM Air Traffic Flow and Capacity Management. 7

ATM Air Traffic Management. 7, 9, 10

ATS Air Traffic Service. 7, 8

BASE Basic Availability, Soft-state, Eventual consistency. 49, 50, 52

CFMU Control Flow Management. 10

CTOT Calculated Take-Off Time. 10

ET Extra Trees Regression. 18

GBM Gradient Boosting Machine. 17

GPS Global Positioning System. 13

IFPS Integrated Flight Plan Processing System. 8, 10

IFR Intrument Flight Rules. 8

ILS Instrument Landing System. 12

IMM Interacting Multiple Model. 20

KNN K-Nearest Neighbors. 17

LSTM Long Short-Term Memory. 17

MSE Mean Square Error. 74

NOAA National Oceanic and Atmospheric Administration. 16

NOTAM Notification to Airmen. 9

OFP Official Flight Plan. 8

RF Random Forest Regresssion. 17

RMSE Root Mean Square Error. 18

RPL Repetitive Flight Plan. 9

RTA Required Time of Arrival. 68, 69, 71, 76, 77, 80

SESAR Single European Sky ATM Research. 16

SID Standard instrument departure. 11

SVM Support Vector Machine. 19

SVR Support Vector Regression. 17

TAF Terminal Aerodrome Forecast. 9

TICK Telegraph, InfluxDB, Chronograph, Kapacitor. 54, 64

VOR Very High Frequency Omnidirectional Range. 12

WX Weather Document. 9