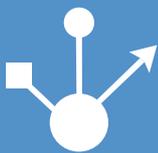


Buscador de vuelos con predicción de retrasos



Máster Universitario en Big Data
Management, Technologies and
Analytics

Trabajo Fin de Máster

Autores:

Raúl Bautista Erustes

Adria Canton Aynes

Adil Ziani El Ali

Tutores:

Alberto Gutiérrez

Pedro Delicado

Liam Patton



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Buscador de vuelos con predicción de retrasos

Autores

Raúl Bautista Erustes

Adria Canton Aynes

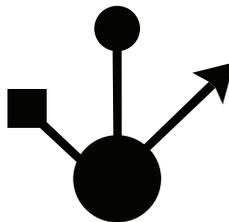
Adil Ziani El Ali

Tutores

Alberto Gutiérrez - Prototyping

Pedro Delicado - Technical

Liam Patton - Business



Máster Universitario en Big Data Management, Technologies and Analytics



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Barcelona, Octubre 2019

Abstract

Our main purpose in this project is to provide a searcher of flights engine to potential customers with basic functionalities such that, given certain preferences by the customer, find the flights that fits best into such preferences. This adjustable parameters are mainly three; price, flight time and flight delay, showing up flight delay as the main novelty, this parameter will be predicted using Machine Learning techniques on historical data from various sources.

Once registered, our customers would be classified in two groups: General users and Business Users, having each of them different options available, for the former finding/booking flights and keep them or the possibility to meet other users with similar features shared, and for the latter, the possibility to plan from a master user, the management of a business trip for all members related. There is also the possibility of car renting or accommodation for both and unregistered customers.

Agradecimientos

Agradecer el trabajo realizado por Pedro Pernías Peco en su plantilla de “tfg” que se puede ver en <https://github.com/lcg51/tfg>. Plantilla \LaTeX de la que hemos partido para adaptar y crear este documento.

También a las empresas **FlightStats**¹, **OpenFlights**² y **OpenWeathermap**³ por ofrecer acceso a versiones de evaluación de sus APIs que hemos usado en el prototipo.

A la Empresa **Sopra Steria Group**⁴, por financiar parte de la matrícula a dicho máster para el alumno Adil Ziani.

Agradecer a los tutores sus observaciones e ideas.

¹<https://developer.flightstats.com>

²<https://openflights.org>

³<https://openweathermap.org>

⁴<https://www.soprasteria.es/es>

Definiciones y acrónimos

- **Accuracy.** Se denomina 'accuracy' o exactitud a la siguiente métrica que nos aporta una "idea" de lo exacto que es un modelo:

$$Accuracy = \frac{\text{número de predicciones correctas}}{\text{número total de predicciones}}$$

- **Recall.** Se denomina 'recall' de una clase, a la siguiente métrica que nos aporta una "idea" de lo exacto que es un modelo en cada clase:

$$Recall_{clase(i)} = \frac{\text{número de predicciones correctas en la clase(i)}}{\text{número total de predicciones en la clase(i)}}$$

- **Schedule.** Denominación, en la API de flightstats, de un vuelo directo, susceptible de ser escala en una ruta que lo contiene (o 'connection').
- **Connection.** Es la denominación de un vuelo entre un origen y un destino, pudiendo contener múltiples escalas.
- **Carrier.** Denominación en la API de flightstats de una aerolínea.
- **Codeshare.** Denominación en la API de flightstats a la gestión alternativa de un vuelo por parte de otras aerolíneas. Por ejemplo, un vuelo operado por Iberia puede tener como codeshare un vuelo operado por Vueling, con un número de vuelo diferente. Pero en definitiva, es la gestión de los billetes del mismo vuelo y el mismo avión físico.

Índice general

Abstract	iii
Definiciones y acrónimos	v
1. Introducción	1
2. Business Analysis	5
2.1. Definición de Proyecto	5
2.1.1. Modelo de negocio CANVAS	6
2.1.2. Análisis de situación DAFO	7
2.1.3. Fuerzas de Porter & Estrategia de Negocio	7
2.2. Budgeting	8
2.3. Aspectos legales	13
3. Technical report	15
3.1. Módulo Data Analytics	16
3.1.1. Datos estáticos	16
3.1.2. RouteStatus.jar	18
3.1.3. RouteRatings.jar	22
3.1.4. AirportWeather.jar	24
3.1.5. Dataset.jar	26
3.1.6. Model.jar	29
3.1.7. Sobre la arquitectura de datos	30
3.2. Módulo búsqueda vuelos	32
3.2.1. Datalake	32
3.2.2. Parámetros	33
3.2.3. Algoritmo	35
3.3. Módulo búsqueda precios	43
3.3.1. Scraper.py	44
3.3.2. GetPrices.py	46
3.3.3. JoinFile.py	48
3.3.4. Herramientas principales	50
4. Data Analysis	51
4.1. Introducción	51
4.2. Data set, desde origen del dato hasta su explotación	53
4.3. Preprocesamiento	55
4.4. Técnicas y algoritmos	57
4.5. Resultados y conclusiones	60
5. Demo	62

6. Conclusiones	66
Bibliografía	68
A. Anexo I - Módulo Data Analytics	69
B. Anexo II - Módulo búsqueda de vuelos	76
C. Anexo III - Sobre análisis	79

1. Introducción

Si lo llego a saber, hubiera aprovechado la tarde y aplazado mi vuelo para salir al día siguiente. Esto es lo que nos sucede a veces; presentarse con dos horas de antelación al aeropuerto y esperar incluso más por retrasos sufridos en nuestro vuelo. ¿Podríamos tener una previsión de ese retraso? ¿O al menos saber con cierta certeza si nuestro vuelo va a sufrir un retraso o no? Si es así, quizás mejor buscar otro vuelo o dejarlo para otro día. Lo importante es evitar perder esa tarde en el aeropuerto. Además, OnTime, permite reducir costes en tiempo y dinero en su viaje, es más, le permite ponderar dichos parámetros según sus expectativas.

A partir de la situación anterior surge la primera semilla de la idea que hay detrás de OnTime, buscar vuelos con predicción de posibles retrasos. Teniendo esa idea en mente, el equipo de OnTime, se enmarca en analizar una idea de negocio más ambiciosa que la simple búsqueda de vuelos, pues en el sector de los viajes y su organización existe un alto potencial de mercado.

Una muestra del comportamiento del mercado es la siguiente noticia:



Figura 1.1: Noticia que muestra una comparativa entre ejercicio 2016 y 2017 de las agencias de viajes online. Fuente Nexotur

De la anterior noticia, y de otras similares, podemos ver que las grandes empresas del sector como Booking Holdings, Skyscanner, Expedia Group, Amadeus, FlightStatus han ido aumentando su facturación en porcentajes relevantes. Lo anterior nos sirve al menos para sacar dos conclusiones: una es que el mercado crece y la otra es que la competencia es enorme,

pero como dijo Jeff Bezos¹ “Obsesiónate con los clientes y no con la competencia”.

OnTime, no únicamente pretende ser un buscador de vuelos de referencia, sino mucho más como veremos mas adelante. Pero le adelantamos que uno de nuestros objetivos es facilitar la organización de viajes a las empresas:



Figura 1.2: Noticia que trata las necesidades de las empresas de contar con un sistema que agilice el proceso de viajes por trabajo. Fuente:hosteltur.com

Y también, como sabemos que viajar es bueno para su salud, y más si está acompañado, OnTime permitirá que los viajeros interactúen entre sí:

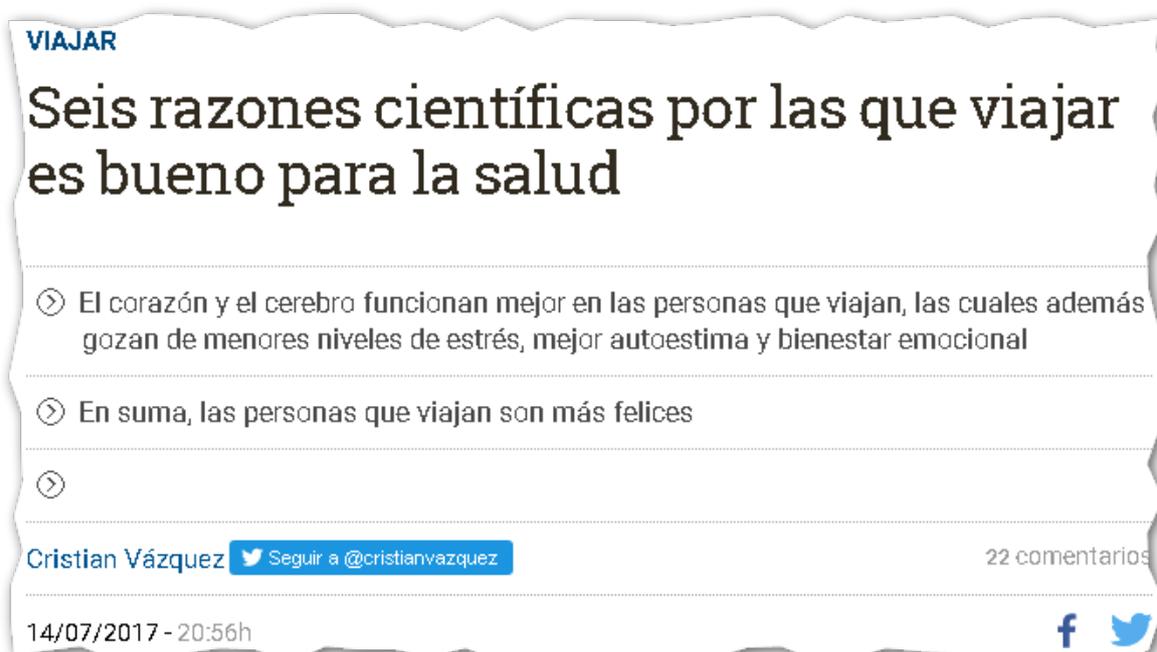


Figura 1.3: Extracción de noticia que trata los beneficios de viajar para la salud. Fuente:eldiario.es

¹Director Ejecutivo de Amazon.

Definición del proyecto

OnTime, como idea de negocio, se compone de 3 módulos:

Módulo 1. Buscador de vuelos

Se desarrolla un buscador de vuelos en el que dado un origen, un destino y una fecha, el usuario define además un peso a cada variable:

- **Precio:** cantidad monetaria total de su trayecto.
- **Tiempo:** duración total de su trayecto.
- **Retraso:** variable predicha por el modelo, que predice el posible retraso del vuelo.

El resultado es el conjunto de rutas aéreas en el orden que mejor se ajusta a las expectativas del usuario, como por ejemplo el orden ponderado según los pesos anteriores.

Este primer módulo, es el que se desarrollará para un prototipo que sirva de entrega a dicho máster. **En el prototipo, el usuario podrá realizar una verdadera compra redirigiéndose a la web que le mostraremos en la aplicación.**

Módulo 2. OnTime como agencia de viajes online

Aquí OnTime pretende ofrecer un servicio completo a los viajeros, tanto los viajes por ocio, como los viajes por trabajo. Por tanto se ofrece la posibilidad de crear una cuenta en la plataforma, donde existen dos perfiles de usuario:

- **Perfil General.** Una vez identificado en la plataforma, podrá organizar sus viajes. Reservar vuelos, trenes, hoteles y alquilar coche, con la garantía de nuestros algoritmos que le ofrecen las rutas que más se ajustan a sus expectativas. Si lo desea, puede activar la opción de *Interactuar* y entonces conocer a usuarios que piensan hacer el mismo viaje con intereses similares. Los usuarios entre ellos pueden organizar quedadas, compartir gastos, intercambiar recomendaciones o simplemente dejar reseñas/observaciones/advertencias sobre lugares que hayan visitado en esa ruta.
 - **Perfil Business.** La plataforma es distinta para este tipo de usuarios, por lo general son *assistant* de empresas quienes crean un perfil manager, que incorpora todos los correos corporativos de sus colaboradores en la empresa. Un colaborador solicita un viaje por trabajo, la plataforma busca rutas, alojamiento, coches (según las variables introducidas por el colaborador) y ofrece un listado de posibilidades que son enviadas a la *assistant* para aceptar ruta (en caso de no activar opción *aceptar automáticamente ruta escogida por colaborador*). Y entonces, se reserva el viaje completo del colaborador, teniendo a su disposición toda la información en la plataforma de manera organizada y simple. La cuenta business ofrece diversas ventajas y configuraciones; la *assistant* puede limitar presupuesto del viaje en general, o por partes (vuelos inferiores a 120 euros, si la ruta es de A a B por ejemplo). Digamos, que cada empresa puede fijar sus políticas, y rutas válidas dentro del abanico de configuraciones que ofrece OnTimey que tendrá en consideración a la hora de organizar viajes.
-

Módulo 3. OnTime ofreciendo servicios de consultoría de Data Analytics

Una vez que el proyecto esté estable, y aprovechando uno de los valores fundamentales de la marca, a saber, la investigación en el análisis de datos y las tecnologías Big Data, OnTime ofrece el servicio de consultoría de analítica de datos, acompañando a sus clientes en la toma de decisiones explotando los datos de los que disponen, así como ver con qué otras empresas pueden cruzar sus datos para enriquecer ambas partes su negocio. Y siempre, sin que ello suponga un riesgo para los datos de sus clientes finales.

Este tercer módulo, no se incluirá en el análisis de negocio.

Estructura del documento

En el capítulo 2, hay un análisis detallado de la idea de negocio, donde se estudia la viabilidad de los Módulos 1 y 2. Se muestran el análisis de negocio CANVAS, el análisis de situación DAFO así como las Fuerzas de Porter y Estrategias de Negocio. El capítulo finaliza con un estudio presupuestario.

En el capítulo 3, encontraremos todos los detalles y análisis funcionales y técnicos de los componentes del módulo 1. El capítulo se divide en tres partes:

- Módulo Data Analytics 3.1. Aquí se exponen todos los detalles referentes a la parte de Data Analytics. El flujo de datos, los componentes y los resultados de los modelos de Machine Learning.
- Módulo Búsqueda de vuelos 3.2. En él se explica el “core” de la aplicación, la utilidad encargada de buscar los vuelos y ofrecerlos al usuario, intercatuando con los otros dos módulos.
- Módulo Búsqueda de precios 3.3. Parte donde se explica el módulo que se encarga de buscar precios a los vuelos hallados por el módulo anterior, así como ofrecer la posibilidad de realizar la compra, ofreciendo el link para reservar.

En cada módulo, se incluyen estadísticas de rendimientos de cada componente y el porqué de las decisiones funcionales y técnicas tomadas a la hora de desarrollar cada programa.

En el capítulo 4, veremos la parte de análisis de datos propiamente dicha; aunque este capítulo iría cronológicamente antes que el Módulo Data Analytics 3.1, preferimos mostrarlo posteriormente partir del conocimiento de los datos de los que disponemos de cómo han sido obtenidos.

En el capítulo 5 mostraremos el flujo completo, desde que introducimos los parámetros de entrada, hasta poder reservar nuestro vuelo.

Finalmente, en el último capítulo 6, veremos las conclusiones del trabajo, así como las posibles mejoras.

En los anexos podrán encontrar detalles técnicos que resultan de utilidad y aclaratorios para el contenido del capítulo *Technical report*.

2. Business Analysis

En este capítulo se efectúa un análisis del negocio para “Buscador de vuelos con predicción de retrasos”, identificando las necesidades del mismo y cómo abordarlas.

Realizar un análisis de negocio ayuda a determinar una estrategia, evaluar las acciones y tener un plan de acción a medio/largo plazo que sirva de guía interna para la empresa/negocio en su trayectoria al éxito. Un elemento clave en esta tarea es examinar el entorno y sectores afines a nuestro negocio para tener una mejor visibilidad e identificar las posibles amenazas y problemas que el negocio puede tener.

Se realiza un análisis de las acciones y estrategias, se identifican problemas y posibles soluciones, cómo abordar el día a día del negocio, así como identificar aquellas acciones que no aportan beneficio y no ayudan hacia el cumplimiento de los objetivos.

2.1. Definición de Proyecto

Como primer paso en Business Analysis, debemos definir cuales son nuestra **Misión, visión y Valores**, lo que nos ayudará a definir factores estratégicos, organizativos y psicológicos de la estrategia empresarial en el medio y largo plazo.

- **Misión.** Dar la posibilidad al usuario de organizar y optimizar su tiempo a la hora de realizar un viaje de la mejor manera posible en función de sus prioridades.
- **Visión.** Ser un buscador y organizador de viajes de referencia en el mercado tanto para el usuario general como para las empresas. Si busca organizar un viaje de ocio o de trabajo de manera fácil, cómoda y segura, tiene OnTime.
- **Valores.** **Igualdad** de género y procedencia, **responsabilidad, trabajo en equipo**, creemos en la **innovación** y la **creatividad** en el proyecto. Integridad con el cliente y los empleados, confidencialidad acerca de los datos del cliente y los empleados.

2.1.1. Modelo de negocio CANVAS

<u>Colaboradores clave</u> <ul style="list-style-type: none"> • Compañías aéreas • Compañías de renting • Servicios de alojamiento • Servicios de hosting • Compañías flight data • Compañía seguros 	<u>Actividades clave</u> <ul style="list-style-type: none"> • Machine Learning • Gestion Flujos de Datos • Gestion usuario Business <u>Recursos clave</u> <ul style="list-style-type: none"> • Acceso datos necesarios • Cluster • Servidor web 	<u>Propuesta de valor</u> <ul style="list-style-type: none"> • Usuario general: buscar vuelos acorde a sus expectativas y prioridades (Tiempo, Retraso y dinero) • Usuario business: además de beneficios usuario general, gestion de viajes por trabajo. 	<u>Relaciones con clientes</u> <ul style="list-style-type: none"> • Servicio general y Servicio business (incluyendo soporte y atención al cliente) <u>Canales</u> <ul style="list-style-type: none"> • Internet 	<u>Segmentos de clientes</u> <ul style="list-style-type: none"> • Viajeros: universitarios, parejas sin hijos, trabajadores cualificados. • Empresas nacionales/internacionales
<u>Costes</u> <ul style="list-style-type: none"> • Mantenimiento de Cluster y Servidores. • Marketing • Compra datos • Salarios, alquiler local 		<u>Ingresos</u> <ul style="list-style-type: none"> • Comisión de vuelo general y comisión de alojamiento/alquiler de coches. • Cuota anual a empresas proporcional a número de cuentas cooperativas + comisión por viaje total. • Banners publicitarios 		

2.1.2. Análisis de situación DAFO

<p style="text-align: center;"><u>Debilidades</u></p> <ul style="list-style-type: none"> • Incertidumbre (y consecuencias) de la predicción • Budget limitado en un inicio. Se necesitan fondos hasta comenzar a generar ingresos. 	<p style="text-align: center;"><u>Amenazas</u></p> <ul style="list-style-type: none"> • Competencia integrada en el mercado
<p style="text-align: center;"><u>Fortalezas</u></p> <ul style="list-style-type: none"> • Innovación en la incorporación de la posibilidad de predicción del retraso • Apoyo a la investigación • Generalización hacia otras redes de transporte 	<p style="text-align: center;"><u>Oportunidades</u></p> <ul style="list-style-type: none"> • Mercado en auge, turismo en auge, viajes de negocios • Confianza/uso creciente en el uso de Machine Learning

2.1.3. Fuerzas de Porter & Estrategia de Negocio

Con el análisis de las cinco fuerzas de Porter, construimos una posible estrategia a medio/largo plazo:

1. Amenazas de entrada

No existen fuertes barreras de entrada. El acceso al cliente es el mismo para todo el sector (internet). Son necesarios "pocos medios" para comenzar. Fácil cumplimiento de las restricciones legales. No obstante, una competencia madura sí es una amenaza, y se considera que se debería procurar, en un tiempo relativamente corto, hacerse con una parte del mercado, para así superar la debilidad de un budgeting inicial limitado y comenzar a generar ingresos lo antes posible. Como estrategia, se abordará la posibilidad de incorporar un chatbot como asistente en el proceso de compra (**sencillez para el usuario**). Para usuarios registrados, ofrecer todos los detalles de un viaje y su organización de manera simple y organizada en la plataforma.

2. Poder de negociación con los clientes

El cliente puede cambiar a la competencia con facilidad aunque su grado de dispersión sea alto. Se ofrece un factor predictivo del retraso como **factor diferenciador** para ayudar a retener al cliente. También se estudiaría la opción de prestación de incentivos a los usuarios registrados para **fidelizear al cliente**.

3. Poder de negociación de los proveedores

El poder de los proveedores es alto. La escasez de proveedores de datos sumado a la condición del proyecto como nueva empresa hace que el poder de negociación de

éstos sea alto. En cuanto a los proveedores de infraestructuras, existe más margen de maniobra, aunque hay un riesgo de integración vertical a tener en cuenta.

4. Productos sustitutivos

Poder sustitutivo importante a favor de los principales buscadores, hacer por tanto **hincapié en los elementos diferenciadores**, la sencillez y eficacia, además de introducir el Machine Learning como parte primordial en el negocio.

5. Competencia directa

Número de empresas en el sector relativamente bajo, pero algunas de ellas son importantes. Mercado en expansión, con lo que es factible captar clientes y parte del mercado en crecimiento. Se necesita promover el sistema de **predicción de retrasos** y la **usabilidad**. Aprovechar también el hecho de que el sistema puede ser generalizable a **otros medios de transporte** (no sólo aplicable a vuelos).

2.2. Budgeting

La idea de negocio consiste en desplegar tres módulos. El primero, un buscador de vuelos que servirá de motor para los sucesivos. Nos llevaría un año de trabajo sin ningún ingreso. Lo previsto sería lanzarlo al mercado después de un año, y comenzar entonces el desarrollo del segundo módulo. El segundo módulo consiste en incorporar búsqueda de hoteles y renting de coches, junto con la incorporación del sistema de gestión de usuarios, tanto las funcionalidades de usuario general como usuario business. Este segundo módulo llevaría un año de trabajo con más personal y se lanzaría al mercado después de dos años de la apertura del negocio. El tercer módulo, consistiría en ofrecer un servicio de consultoría de Data Analytics. No se incluye en este estudio de Budgeting ni Business Analysis en general, no obstante sería interesante su incorporación para aprovechar uno de nuestros valores, el apostar por la investigación en el área de Data Analytics.

Después de elaborar una P&L Profit and Lose model, tenemos tres escenarios. El mejor de ellos supone alcanzar las expectativas sin retrasos. En el segundo, supondría un retraso de seis meses. En el peor escenario se tendría un retraso de 1 año.

El budgeting se realiza para 3 años.

Supuestos:

1. Para los préstamos bancarios se va a considerar un interés fijo del 5% a pagar en 3 años. No se considera interés variable¹
2. Supongamos que logramos llegar a tener en un año el 5% del tráfico de la web de Skyskanner.com y del cual el 10% efectúa una reserva. Es decir, el 0.5% genera ingresos el primer año. El segundo año suponemos subida hasta el 0.8%.²

¹Las cuentas de préstamos e intereses se han realizado usando simuladores de préstamos <https://www.arpem.com/financiacion/calculadoras/>.

²Numero de visitas a la web skyscanner.com en <https://www.similarweb.com/website/skyscanner.com>

3. En relación a la posibilidad de contratar alojamiento, consideramos que el 10% de las personas que hacen reserva de vuelo también reservan alojamiento.
4. Para el alquiler de coches, consideramos que 1% de las personas que reserva el viaje, también reserva un coche.
5. Para los usuarios business, consideramos que estos en volumen pueden llegar a ser el 0.01% de los usuarios “ general”.

Mejor previsión. Sin retrasos

BEST CASE SCENARIO: P&L	2020	2021	2022
TOTAL INCOME	0,00 €	1.862.962,50 €	7.468.188,75 €
<i>Cum.</i>	<i>0,00 €</i>	<i>1.862.962,50 €</i>	<i>9.331.151,25 €</i>
TOTAL EXPENSES	-224.323,80 €	-626.606,18 €	-1.126.776,95 €
<i>Cum.</i>	<i>-224.323,80 €</i>	<i>-850.929,98 €</i>	<i>-1.977.706,93 €</i>
P&L BUSINESS CASE (EBIT)	-224.323,80 €	1.236.356,33 €	6.341.411,80 €
<i>Cum.</i>	<i>-224.323,80 €</i>	<i>1.012.032,53 €</i>	<i>7.353.444,33 €</i>
ROI BT (Return of Investm. Before Tax)	-100%	197%	563%
<i>Acum.</i>	<i>-100%</i>	<i>97%</i>	<i>660%</i>

Tabla 2.1: Mejor escenario



Figura 2.1: Budgeting mejor escenario

Previsión con 6 meses de retraso

AVERAGE CASE SCENARIO:	2020	2021	2022
TOTAL INCOME	0,00 €	1.036.237,50 €	5.959.096,25 €
<i>Cum.</i>	<i>0,00 €</i>	<i>1.036.237,50 €</i>	<i>6.995.333,75 €</i>
TOTAL EXPENSES	-224.323,80 €	-626.606,18 €	-1.126.776,95 €
<i>Cum.</i>	<i>-224.323,80 €</i>	<i>-850.929,98 €</i>	<i>-1.977.706,93 €</i>
P&L BUSINESS CASE (EBIT)	-224.323,80 €	409.631,33 €	4.832.319,30 €
<i>Cum.</i>	<i>-224.323,80 €</i>	<i>185.307,53 €</i>	<i>5.017.626,83 €</i>
ROIBT (Return of Investm. Before Tax)	-100%	65%	429%
<i>Acum.</i>	<i>-100%</i>	<i>-35%</i>	<i>394%</i>

Tabla 2.2: Escenario retrasos 6 meses

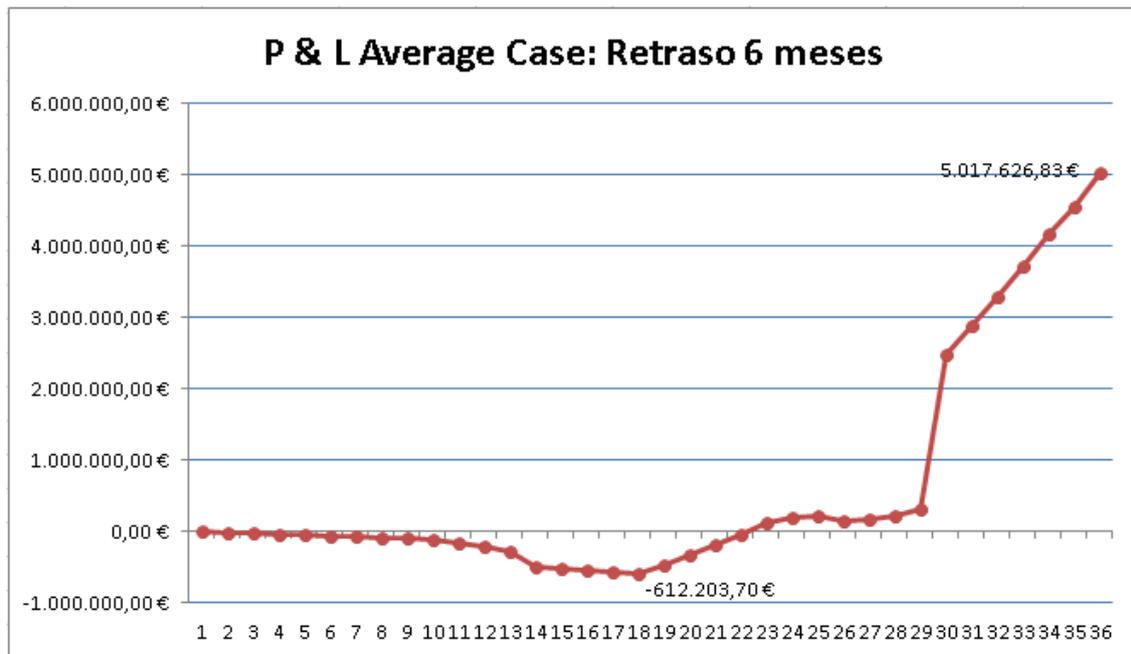


Figura 2.2: Budgeting escenario retraso 6 meses

Previsión con 1 año de retraso

WORST CASE SCENARIO:	2020	2021	2022
TOTAL INCOME	0,00 €	0,00 €	1.862.962,50 €
<i>Cum.</i>	<i>0,00 €</i>	<i>0,00 €</i>	<i>1.862.962,50 €</i>
TOTAL EXPENSES	-224.323,80 €	-626.606,18 €	-1.126.776,95 €
<i>Cum.</i>	<i>-224.323,80 €</i>	<i>-850.929,98 €</i>	<i>-1.977.706,93 €</i>
P&L BUSINESS CASE (EBIT)	-224.323,80 €	-626.606,18 €	736.185,55 €
<i>Cum.</i>	<i>-224.323,80 €</i>	<i>-850.929,98 €</i>	<i>-114.744,43 €</i>
ROI BT (Return of Investm. Before Tax)	-100%	0%	65%
<i>Acum.</i>	<i>-100%</i>	<i>-200%</i>	<i>-135%</i>

Tabla 2.3: Escenario peor caso

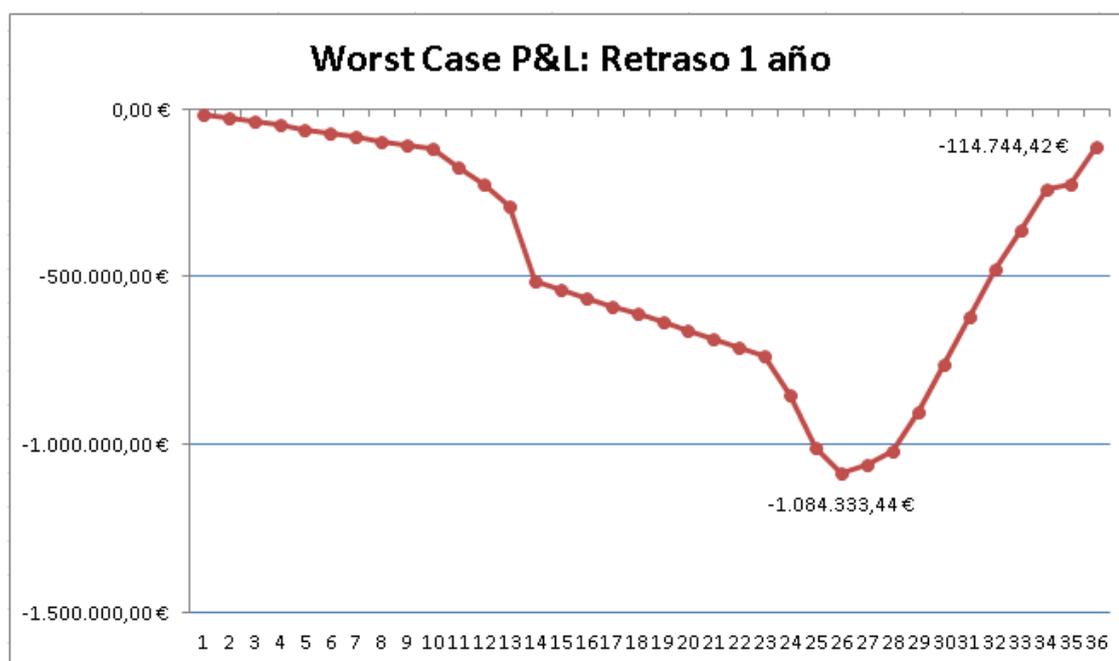


Figura 2.3: Budgeting escenario retraso 1 año

Como observamos en las gráficas, y con unos gastos bastante razonables que podremos consultar en el excel de budgeting, la idea de negocio compromete numéricamente al menos en los dos escenarios favorables. Veamos a continuación el cálculo de préstamos³ y situaciones del negocio según cada escenario:

Peor escenario - retraso de un año en la puesta en marcha del producto:

³Las gráficas muestran las cuentas de la compañía en el caso de tener financiación interna o business angel que acepte como pago parte de la compañía asumiendo riesgos.

Un retraso de un año en la puesta en marcha del producto, provocaría una deuda acumulada de **-114.744 €** al cerrar el tercer año, sumando los intereses la deuda total asciende a **-157.317 €**⁴. Con lo cual, este escenario cerramos el tercer año aun con deudas y dificultades, es insostenible. En definitiva, **el control del tiempo es crucial**.

Average case - retraso de medio año en la puesta en marcha del producto:

En este caso la estrategia es solicitar préstamos pequeños a 3 años como sigue: un préstamo inicial de **-225.000 €** de los cuales -224.323 € irían a gastos del primer año. La cuota a devolver de la deuda sería de -79.600 € y los gastos del negocio de los primeros 6 meses del segundo año ascienden a -387.880 €, con lo cual escogemos un segundo prestamos de **-467.480 €**, de los cuales -79.600 € irían destinados a pagar la cuota del primer préstamos. Al finalizar el segundo año los ingresos totales de compañía serían de +1.036.237 € y los gastos totales de -626.606 € de los cuales parte ya se financió con el segundo préstamo, es decir, estamos en un balance de $(+1.036.237 - 626.606 + 387.880)$ ⁵ = +797.511 € y las cuotas a devolver serían de -79.600 € - 168.132 € = -247.732 € Con lo cual, el segundo año cierra en +246.935 € con todos los gastos pagados y las cuotas de la deuda pagadas. El tercer año es posible un autofinanciamiento cerrando el año con +4.832.320 € de beneficios con cuota a pagar de -247.732 €, es decir, **+4.585.385 €**.

En definitiva, este escenario es rentable.

Best case - sin retraso en la puesta en marcha del producto:

En este caso la estrategia es la misma que la anterior, dos préstamos a 3 años, el primer de **-225.000 €** como el caso anterior. A inicios del segundo año, un segundo préstamos de -354.136 - 79.600 = **-274.536 €** para pagar la cuota del primer préstamo y sufragar los gastos de los primeros 3 meses del segundo año. El segundo año cierra con un beneficio de +1.236.356 € y unos gastos de -626.606 € de los cuales -354.136 € se sufragan con el segundo préstamo. En definitiva, el segundo año cierra con +963.886 € y la cuota a pagar sería de -79.600 - 98.736 = -178.336 €, quedándose en +785.550 €. El tercer año es posible un autofinanciamiento cerrando el año con +6.3411.411 € de beneficios con cuota a pagar de -178.336 €, es decir, **+6.163.075 €**.

Este escenario también resulta rentable.

En cuanto a las fuentes de ingresos, éstas son:

- Módulo 1: 0.03 €/ click banners + 2.99 € por gestión viaje reservado.
- Módulo 2: 0.03 €/ click banners + 2.5% gastos gestión reserva hotel + 2.5% gastos gestión reserva coche + 5% gastos gestión reserva viaje usuario business⁶

⁴Para calcular el interés, se toma la deuda acumulada al finalizar el segundo año, se le aplica un 5%, el resultado se resta al beneficio del tercer año.

⁵Corresponde a ingresos totales - gastos totales + sobras del préstamo 2 al pagar cuota del préstamo 1.

⁶De media, los españoles se gastan 750 Euros en transporte y alojamiento en un viaje de negocios. Fuente https://www.hosteltur.com/124682_empresas-gastan-media-70000-al-ano-viajes-corporativos.html

En cuanto a los gastos, éstos provienen principalmente de:

OnTime	2020	2021	2022
1.0 COSTS (EXPENSES): OPERATIVE	-141.436,00 €	-294.521,00 €	-375.734,00 €
1.a. Labour costs/HR	-107.736,00 €	-257.976,00 €	-334.484,00 €
1.b Suppliers	-3.000,00 €	0,00 €	0,00 €
1.c. Offices/ Others	-23.980,00 €	-29.105,00 €	-33.810,00 €
1.d. Production expenses	-6.720,00 €	-7.440,00 €	-7.440,00 €
2.0 MARKETING (ADVERTISING)	-80.000,00 €	-328.097,38 €	-746.555,15 €
4.0 Expenses on equipment amortization	-2.887,80 €	-3.987,80 €	-4.487,80 €
TOTAL EXPENSES	-224.323,80 €	-626.606,18 €	-1.126.776,95 €
<i>Cum.</i>	<i>-224.323,80 €</i>	<i>-850.929,98 €</i>	<i>-1.977.706,93 €</i>

Tabla 2.4: Resumen de los gastos

2.3. Aspectos legales

En OnTime, será necesario establecer cautelas en los ámbitos siguientes:

1. Técnicas de web scraping.
2. Normativas generales relativas a los reglamentos generales de protección de datos (LOPD y RGPD).

Web scraping

En el prototipo se han utilizado técnicas de web scraping. El web scraping permite extraer información de webs simulando la navegación de un humano, para poder ser procesada o utilizada por terceros. Es una técnica legal, como dicta la sentencia del Tribunal Supremo de 9 de octubre de 2012 número 572/2012 a la demanda de Ryanair contra Atrápalo <https://supremo.vlex.es/vid/desleal-ryanair-atrapalo-as-411388894>.

Únicamente habría que tener en cuenta el no generar situaciones anómalas que pudieran ser asimiladas a algo similar a un ataque de denegación de servicio (DDoS), o apropiarse de material susceptible de ser una violación de derechos de autor.

Pero no es el caso, ya que se accede a información simplemente de precios que es pública.

En cualquier caso, como se ha comentado, el scraping se efectúa en el prototipo para el acceso a los precios existentes de rutas que el usuario pueda demandar. En el proyecto real se accederían a APIs existentes (como la de skyscanner por ejemplo: https://skyscanner.github.io/slate/?_ga=1.104705984.172843296.1446781555#flights-browse-prices que son de pago.

LOPD, RGPD

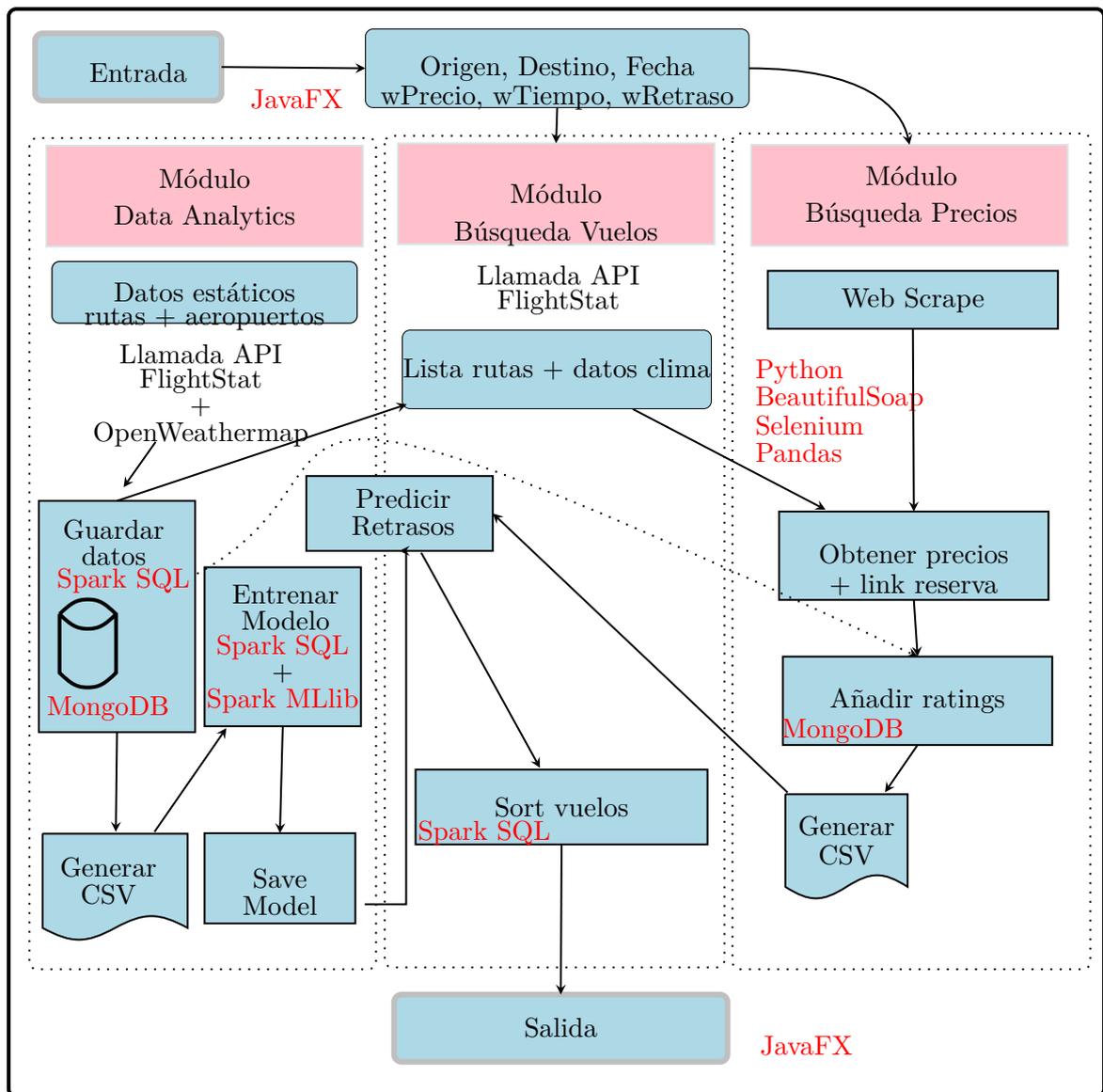
En relación a la creación de cuentas de usuario para el Módulo 2 de OnTime, deben ser tenidas en cuenta tanto las normativas a la Ley Orgánica de Protección de datos (LOPD, [https://es.wikipedia.org/wiki/Ley_Org%C3%A1nica_de_Protecci%C3%B3n_de_Datos_de_Car%C3%A1cter_Personal_\(Espa%C3%B1a\)](https://es.wikipedia.org/wiki/Ley_Org%C3%A1nica_de_Protecci%C3%B3n_de_Datos_de_Car%C3%A1cter_Personal_(Espa%C3%B1a))), como el nuevo reglamento europeo RGPD <https://rgpd.es/>.

Destacar en este sentido la mejora que para el usuario tiene RGPD en varios ámbitos, de los que destacamos tres:

- La portabilidad de los datos del usuario (posibilidad de transferir los datos a otra empresa)
- El derecho al olvido (eliminación de datos obsoletos de internet)
- La limitación en el tratamiento de los mismos (que sean usados para determinados fines).

OnTime deberá estar preparado para ofrecer a sus usuarios las anteriores demandas, para ello cuenta de un presupuesto en temas jurídicos para su asesoramiento.

3. Technical report



3.1. Módulo Data Analytics

El módulo “Data Analytics” se encarga de guardar datos de vuelos y del clima de los aeropuertos (o las localidades más próximas) para su posterior explotación en el entrenamiento de modelos de predicción de los retrasos. También se encarga de ofrecer datos del clima para los días futuros y ratings de rutas aéreas actualizadas que son necesarias para el módulo “Búsqueda de Vuelos”. Así como del entrenamiento de modelos.

Dicho bloque lo organizaremos en forma de sub-bloques, donde cada bloque corresponde a una parte del workflow. Nos detenemos en cada parte para explicar su utilidad así como el análisis funcional/técnico de los programas.

3.1.1. Datos estáticos

La API “Flight Status by Route”¹ que nos ofrece datos de vuelos, necesita una ruta ejemplo “BCN-MAD” y una fecha de salida de vuelo “2019-10-22”. La API “Ratings API”² que nos ofrece estadísticas de los vuelos de una ruta, necesita como entrada una ruta tipo “BCN-MAD”. Y finalmente, la API “Weather API”³ que nos devuelve datos del clima, necesita de las coordenadas geográficas del aeropuerto.

Entonces, necesitamos proporcionar estos datos de entrada, que forman parte de lo que hemos llamado **Datos estáticos**:

La API “Airports API”⁴ nos ofrece un json con todos los aeropuertos activos, donde cada documento embebido, contiene información del aeropuerto. Véase un ejemplo en el **anexo 1 (A)**. De aquí podemos obtener un fichero con aeropuertos y sus coordenadas geográficas que sirva de fichero en entrada para la API “Weather API” cuyo formato lo tenemos en **anexo 1 (A)**.

En este punto tenemos un fichero formato csv de entrada para el Batch de descarga de datos climáticos. Pero, en un prototipo no podemos tratar todos los aeropuertos existentes ni todas las rutas posibles, necesitamos filtrar. Antes de nada, unas observaciones sobre la cantidad de datos que existen en la realidad:

- **Aeropuertos activos:** 16.214 identificados por IATA “International Air Transport Association”, con lo cual, habrá más aeropuertos activos que esa cifra, pero seguramente no mucho más ya que en IATA se encuentran registradas la mayoría de las aerolíneas.
- **Aerolíneas activas:** 1.754 aerolíneas/compañías que ofrecen servicios de vuelos.
- **Rutas aéreas:** No podríamos considerar todas las rutas posibles, sería de orden cuadrático del número de aeropuertos y de las cuales, muchas rutas no tendrían conexión directa. Dicha información no es fácil de conseguir dado que no fue posible hallar un

¹<https://developer.flightstats.com/api-docs/flightstatus/v2/route>

²<https://developer.flightstats.com/api-docs/ratings/v1>

³<https://openweathermap.org/api>

⁴<https://developer.flightstats.com/api-docs/airports/v1>

registro de rutas actualizado. El que tenemos fue creado por **Openflights** que contiene 67.663 rutas entre 3.321 aeropuertos con 548 aerolíneas operando.



Figura 3.1: Rutas aéreas 2014. Fuente:<https://openflights.org/data.html#route>

Con lo cual, usando las rutas de **Openflights** abarcamos gran parte del planeta como se observa en la image anterior. Pero aun así, manejar 16 mil rutas, por limitaciones de acceso a los datos, no será posible. A saber, que la versión de pruebas de flighStats, nos ofrece 20.000 llamadas al mes, con lo cual, reduciremos el número de rutas y por ende el número de aeropuertos.

La estrategia que hemos seguido para filtrar es la siguiente:

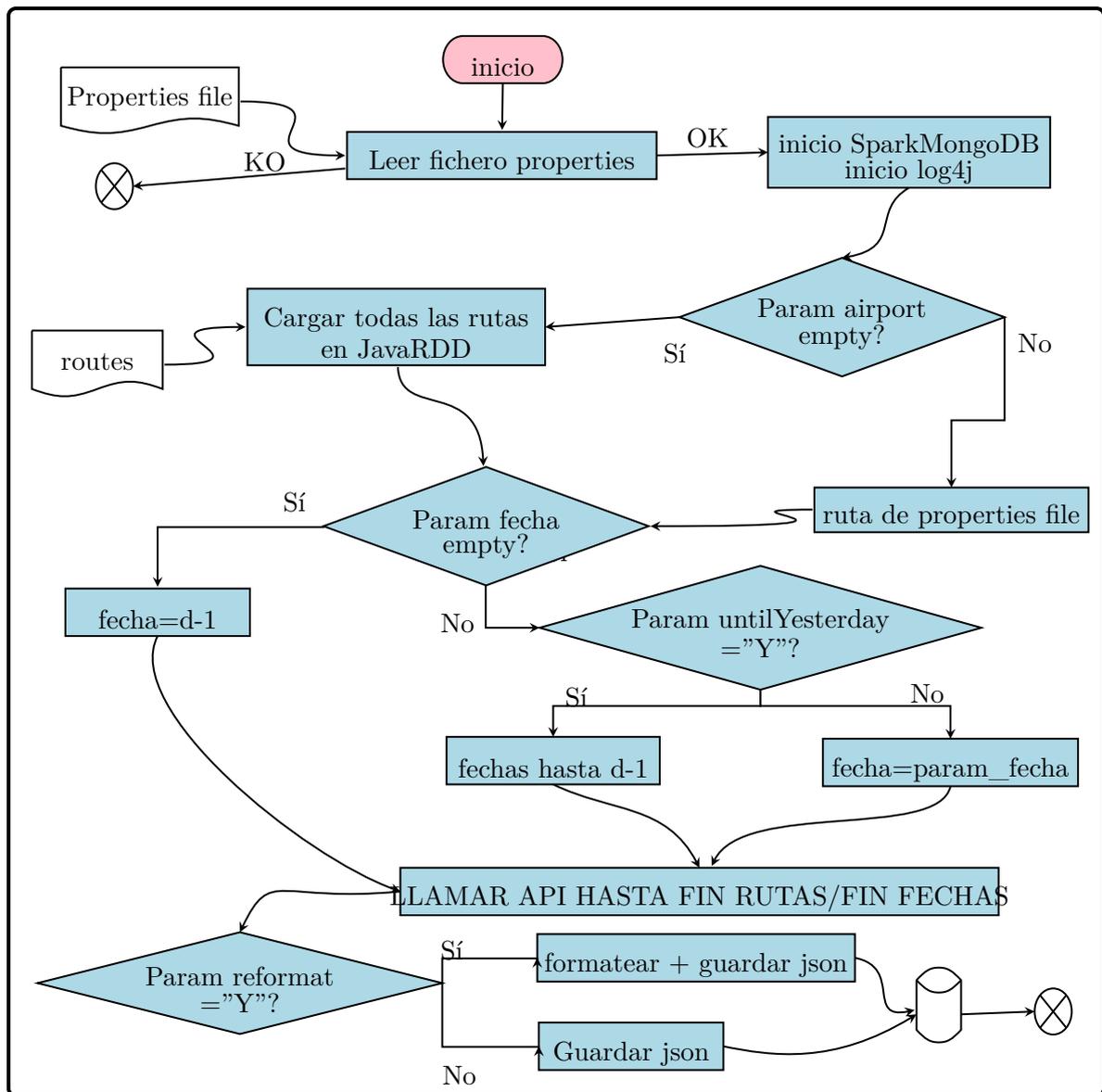
1. Considerar los 55 aeropuertos más transitados según wikipedia 2015, https://es.wikipedia.org/wiki/Anexo:Aeropuertos_por_tr%C3%A1fico_de_pasajeros
2. Considerar toda ruta, del fichero rutas que nos ofrece Openflights, donde el aeropuerto salida o llegada aparece en la lista de los 55 aeropuertos.
3. Crear un fichero de aeropuertos resultado, los que componen las rutas escogidas en el apartado anterior.
4. Crear el fichero coordenadas para dicha lista de aeropuertos.
5. Hacer una pasada por la API “Weather API”, y suprimir aquellos aeropuertos de los que no disponemos de datos de clima.
6. Volver a filtrar fichero rutas, eliminando aquéllas que en aeropuerto salida, contiene algún aeropuerto sin datos de clima.
7. Crear un fichero rutas, como el que aparece en **anexo 1 (A)**

Así, y por medio de programas java, tenemos los ficheros en entrada (**Datos estáticos**) para los batches que acceden a las APIs, con un volumen de:

- Rutas: 10.745
- Aeropuertos: 1.386

También, y como veremos en el módulo de búsqueda precios, hemos necesitado disponer de una lista de aerolíneas con su nombre y código iata. Dicha lista es un json creado a partir de la lista <https://www.kayak.com/airlines> que forman las aerolíneas a los que tiene acceso Kayak y que forman un total de 556 aerolíneas. También tenemos un json de flightStats con 1754 aerolíneas. Dicho json hace de diccionario entre nombre compañía y su código iata correspondiente.

3.1.2. RouteStatus.jar



Como dijimos anteriormente, la API “Flight Status by Route” nos ofrece información sobre los vuelos. Los detalles sobre la respuesta de dicha API, pueden hallarse en (Cirium, s.f.-b) y una muestra del json que devuelve, por ejemplo para la ruta “AAE-ALG” el día “2019-07-22” puede hallarse en **anexo 1 (A)**, no obstante, más adelante comentaremos algunas de las variables importantes.

Para acceder a esta información y almacenarla, disponemos de dos batch en Java: routeStatus.jar y routeStatusSpark.jar, ambos hacen lo mismo, únicamente en el segundo usamos una conexión distribuida.

Requisitos:

El programa necesita:

1. Fichero de properties **anexo 1 (A)**.
2. Fichero de rutas (A).
3. Claves de acceso a la API.
4. Sesión de MongoDB iniciada.

Resultado:

Documentos JSON con información sobre los vuelos (A)

La lógica del programa, nos permite:

- **Bajar los datos de una ruta y una fecha dada.** Los parámetros departureAirport, arrivalAirport, year, month, day bien introducidos. El parámetro reformat=Y si queremos guardar documentos reformateados según nuestras necesidades, si no, dejar vacío. El parámetro untilYesterday debe ser distinto de Y.
 - **Bajar los datos de una ruta y la fecha de ayer.** Los parámetros departureAirport, arrivalAirport bien introducidos. Alguno de los parámetros year, month, day a valor -1. El parámetro reformat=Y si queremos guardar documentos reformateados según nuestras necesidades, si no, dejar vacío. El parámetro untilYesterday debe ser distinto de Y.
 - **Bajar los datos de una ruta desde la fecha indicada hasta la fecha de ayer.** Los parámetros departureAirport, arrivalAirport, year, month, day bien introducidos. El parámetro reformat=Y si queremos guardar documentos reformateados según nuestras necesidades, si no, dejar vacío. El parámetro untilYesterday debe ser igual a Y.
 - **Bajar los datos de toda una lista de rutas dada una fecha.** Los parámetros departureAirport, arrivalAirport debe alguno de ellos estar vacío. Year, month, day bien introducidos. El parámetro reformat=Y si queremos guardar documentos reformateados según nuestras necesidades, si no, dejar vacío. El parámetro untilYesterday debe ser distinto de Y. El parámetro data_path debe apuntar al fichero de rutas.
-

- **Bajar los datos de toda una lista de rutas desde una fecha indicada hasta ayer.** Los parámetros `departureAirport`, `arrivalAirport` debe alguno de ellos estar vacío. `Year`, `month`, `day` bien introducidos. El parámetro `reformat=Y` si queremos guardar documentos reformateados según nuestras necesidades, si no, dejar vacío. El parámetro `untilYesterday` debe ser igual a `Y`. El parámetro `data_path` debe apuntar al fichero de rutas.
- **Bajar todos los datos de una lista de rutas para el día de ayer.** Los parámetros `departureAirport`, `arrivalAirport` debe alguno de ellos estar vacío. `Year`, `month`, `day` al menos uno de ellos a `-1`. El parámetro `reformat=Y` si queremos guardar documentos reformateados según nuestras necesidades, si no, dejar vacío. El parámetro `data_path` debe apuntar al fichero de rutas.

Como vemos, el programa nos permite guardar los datos del pasado en caso de olvidar lanzarlo un día para recuperar la información del día anterior. Además, dispone de una gestión de log (log4j) que nos permite saber el último lanzamiento en qué día ha sido y si acabó correctamente. Pero lo ideal sería lanzarlo cada día a una determinada hora preferiblemente a partir de las 10:00h para asegurarse de tener toda la información del día anterior actualizada a nivel mundial, y escoger la opción de lista de rutas sin fijar fecha y sin especificar `untilYesterday` a `Yes`. Cabe destacar que en caso de error, el programa sin versión de Spark es capaz de detectar si la información ya existe en MongoDB y en caso afirmativo saltar la ruta en cuestión.

En el documento de respuesta, tenemos un array de documentos “flightStatuses” donde cada documento corresponde a un vuelo en la ruta, tenemos información sobre la compañía, numero vuelo, fecha salida, fecha llegada, hora salida, hora llegada (tanto las publicadas como las reales), delays en salida, delays en llegada, tipo vuelo, tipo aeronave, terminales, duraciones de las distintas fases por las que pasa un vuelo entre otras variables. Además, en la versión de pago, existen más datos interesantes, como los incidentes que haya podido sufrir el vuelo (ésto para conocer causas de retrasos sería interesante).

Cabe destacar que la cantidad de datos es verdaderamente Big Data. Usando cifras redondeadas, podemos suponer que tenemos 67.000 rutas y de media 3 vuelos por cada ruta, luego cada día se pueden registrar entorno a 200.000 vuelos. Si queremos por ejemplo hacer un análisis por series temporales donde necesitemos al menos recolectar un año de información, entonces el análisis se haría sobre $365 * 200.000 = 730.000.000$ vuelos que se traduce en cantidad de datos a gestionar. Le dejamos el siguiente enlace donde podrá comprobar dichas cantidades con más exactitud <https://www.flightradar24.com/data/statistics>.

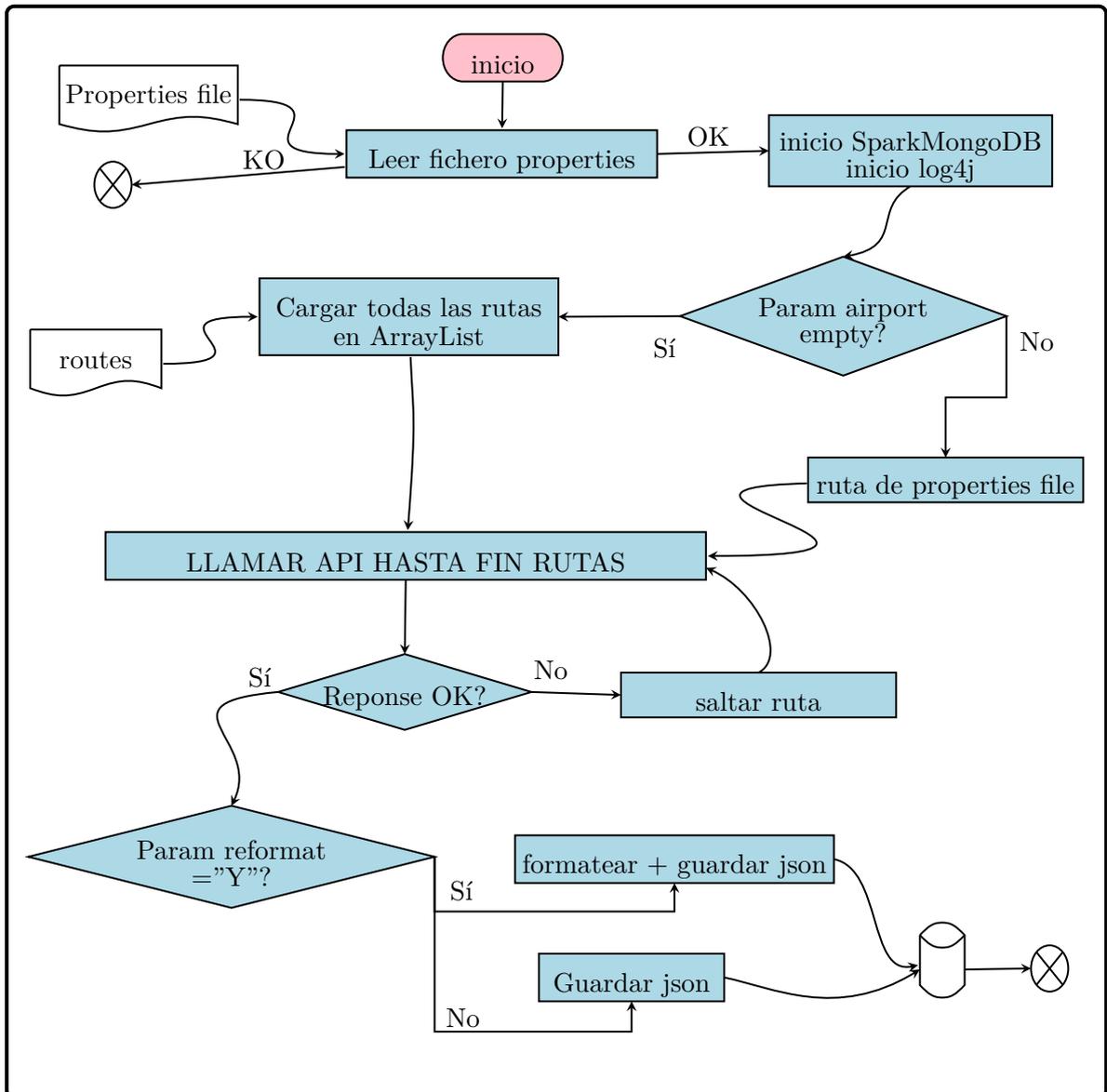
Testing

La versión normal sin paralelismo de Spark, ni Threads en Java, requiere de *29.1min* aproximadamente para acceder a la información de 12.169 rutas. Hacer las llamadas a la API por cada ruta, procesar la respuesta y guardarla en mongoDB. La versión Spark, con dos procesadores, baja el tiempo a *12.3min*. Con procesador intel i5 2.40GHz x2. Seguramente, aunque el rendimiento no bajaría de forma lineal con número nodos (procesadores en local), la manera que tiene Spark de particionar el fichero de entrada por los nodos, y ser un algoritmo independiente sin necesidad de shuffling, en computadoras de más núcleos o en Clusters, el

rendimiento sería mucho mejor. En definitiva, el batch es escalable, bastaría que la API responda en threads o múltiples llamadas de la misma clave al mismo tiempo, cosa que soporta sin problemas.

En cuanto al modelado de la arquitectura de los datos, los documentos se guardan en una colección MongoDB. La clave es del tipo *_id : departureAirport + arrivalAirport + launchDate + LaunchHour* (que nos permite búsquedas más sencillas a la hora de calificar& testear los datos y programas). En caso de solicitar reformatear el documento, se suprimen algunos datos de la API que no consideramos necesario guardar, así ganamos en espacio. Pero por lo general el acceso a dicha colección es un acceso secuencial para recorrer todos los documentos, por lo que no es necesario una arquitectura mas sofisticada de acceso rápido por id o índices. No obstante, en caso de tener acceso a muchos datos y durante muchos años, a la hora de entrenar los modelos quizás sólo nos interesan los datos posteriores a cierta fecha, en tal caso una ID por unix epoch permitiría un acceso más rápido usando los índices B-Tree de MongoDB. Cabe recordar aquí que una clave de este tipo favorece la lectura pero puede perjudicar la escritura si los timestamp están muy próximos, pero al ser la escritura un proceso batch se puede permitir el perder algo de tiempo en eficiencia a favor de búsqueda rápida.

3.1.3. RouteRatings.jar



RouteRatings.jar nos permite obtener estadísticas actualizadas de los vuelos accediendo por rutas, es decir, dada una ruta por ejemplo “BCN-MAD” obtenemos sub-documentos embebidos con información de los vuelos de dicha ruta.

La información que ofrece dicha API puede consultarse en Cirium (s.f.-c) y una muestra del JSON formateado para nuestros intereses puede consultarse en **Anexo I (A)**. La información relevante es la media de retrasos *delayMean*, *allOnTimeCumulative* y *allDelayCumulative*, las dos últimas variables son puntuaciones, en tanto por 1, en cuanto a tener retraso o no que calcula la API para el vuelo en comparación con el resto de vuelos. En cuanto al resto de variables, aunque no tiene ningún uso cara al modelo, podría ser información para el usuario, conocer estadísticas de su vuelo, etc.

Requisitos:

El programa necesita de:

- Fichero de parámetros. A
- Fichero de rutas. A
- Sesión de MongoDB

Resultado:

Insertar documentos en MongoDB, A.

La lógica del programa nos permite:

- **Descargar las estadísticas de los vuelos de una ruta dada**, introducida por fichero de properties. Los aeropuertos han de estar bien informados. También permite insertar el documento tal cual es recibido indicando *reformat* a N, o hacer dos tipos de formateo donde cambia la *_id* y la manera de guardar los documentos.
- **Descargar las estadísticas de un listado de rutas**, en tal caso un aeropuerto ha de estar sin informar. Igualmente, como el caso anterior, dos tipos de estructura de documentos.

En cuanto a la frecuencia de lanzamientos, es suficiente con actualizar la información cada dos o cuatro semanas, con lo cual, y al ser datos acumulativos, se debe borrar la colección y actualizarla por otra nueva mejor que hacer update ya que toda la colección se actualiza.

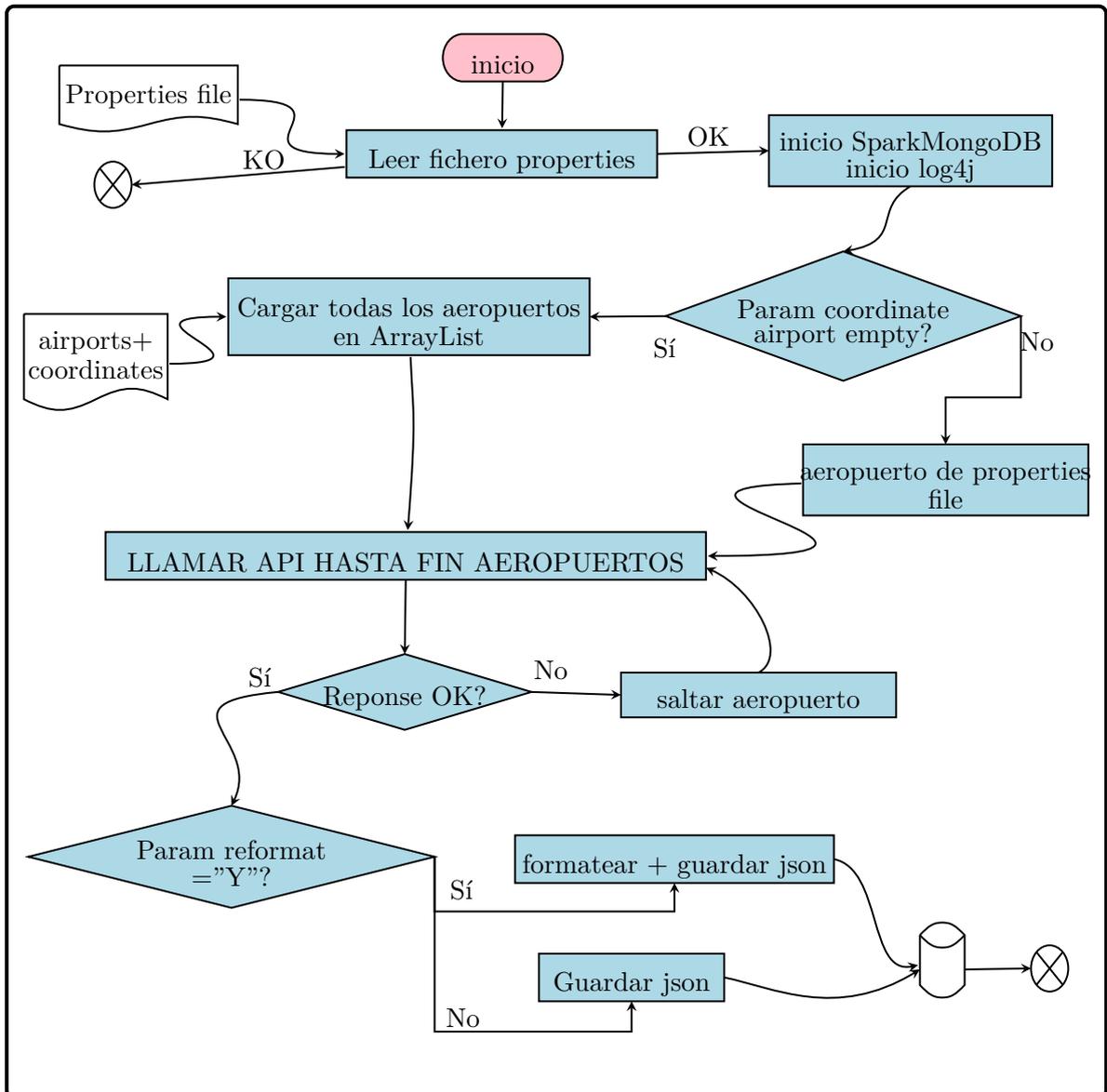
Testing

Igual que con RouteStatus.jar, disponemos de dos versiones, una sin paralización y otra con Spark. En el primer modelo de arquitectura (*Reformat = Y* y *reformat2 = N*), el documento se inserta por ruta con *_id : departureAirport + arrivalAirport + launchDate* y el programa tarda *21min* para descargar y guardar los documentos para 10745 rutas y la versión spark, reduce el tiempo a poco menos de la mitad *9min*.

Esta arquitectura donde cada documento corresponde a una ruta y embebido contiene un array de documentos correspondientes a estadística de vuelos, requiere menos tiempo que la segunda arquitectura que veremos pero no es eficiente a la hora de ir a buscar las estadísticas de un vuelo dado, ya que primero busca por aeropuertos y luego se mira en el array de documentos por el vuelo.

La segunda arquitectura (*Reformat = Y* y *reformat2 = Y*) consiste en guardar los sub-documentos fuera del array como documentos y darles un *_id : departureAirport + arrivalAirport + carrierFScode + numberFlight* (A). De esta manera se gana bastante en eficiencia en los programas que necesitan acceder a routeRatings ya que buscan por *_id*(de hecho, el programa dataset.jar ha pasado de tardar 41 minutos a tan solo 3.75 minutos!!). Aún que a la hora de guardar la información, los tiempos suben a *34min* y *16min* para la versión Spark.

3.1.4. AirportWeather.jar



AirportWeather.jar nos permite tener datos del clima de una zona geográfica usando las coordenadas geográficas de tal lugar. La compañía openweathermap ofrece diversos datos pero la versión gratuita que aquí usamos, nos da cierta información del clima de los próximos 5 días (la versión de pago ofrece hasta un mes de previsiones). Así que, guardamos dicha información de forma regular para ver si podemos sacar partido de los datos del clima a la hora de entrenar modelos. La información que ofrece dicha API puede consultarse en openweather (s.f.) y un ejemplo del json en **Anexo I (A)**.

Requisitos:

- Fichero de parámetros A

- Fichero de aeropuertos con coordenadas A
- Sesión de MongoDB

Resultado:

Datos del clima con 5 días de previsión en la versión gratuita de la API y 30 en la de pago.

La lógica del programa nos permite:

- **Descargar datos del clima para un aeropuerto dado.** Los parámetros longitude, latitude y airport han de estar bien alimentados.
- **Descargar los datos del clima para una lista de aeropuertos.** El parámetro data_path ha de apuntar al fichero de aeropuertos con coordenadas A y al menos algún parámetro del caso anterior ha de estar vacío.

En cuanto a la frecuencia de lanzamientos de dicha aplicación, se automatizan los lanzamientos para lanzar el proceso una vez cada 5 días en caso de este prototipo.

Testing

En este batch, para el prototipo, no se ha desarrollado ninguna versión que haga uso de herramientas de Big Data, ni distribución ni escalabilidad. La razón es que la API en versión gratuita limita las conexiones por minuto a 60 llamadas, con lo cual no merece la pena crear programas sin poder testarlos ni calificar su rendimiento y escalabilidad. Es más, en el código, ralentizamos la ejecución para no bloquear el acceso a la API. Con lo anterior, el programa tarda *29min* en descargar y guardar los datos del clima de los 1535 aeropuertos que usamos en nuestro prototipo.

Cabe destacar que dichos datos, se consultan posteriormente en tiempo real y también a la hora de crear el dataset para entrenar los modelos, con lo cual se ha reflexionado bastante sobre la arquitectura de los documentos, y después de diversas pruebas se han añadido ciertas variables como “dt_local” que es la fecha local del aeropuerto en formato epoch time (lo usará la API connections de búsqueda vuelos que por defecto usa horas locales de aeropuertos), “airport”, “dt_0” y “dt_1” donde airport nos permite identificar más rápido el clima de un aeropuerto dado sin tener que usarla de *_id* ya que se repite. dt_0 es la hora local en epoch time del primer sub-documento y dt_1 la del último sub-documento. Los sub-documentos son datos del clima de una determinada hora, ya que la API devuelve datos de 5 días de previsión en intervalos de 3h.

Entonces, una manera de buscar el documento donde va a existir los datos del aeropuerto y fecha que nos interese es la siguiente: filtrar por aeropuerto + epoch time menor que dt_1, así sabemos que en ese documento ya tendremos una hora que se acerque al menos *1.5h* de nuestra fecha.

Código 3.1: Filtrar airportWeather

```
1 airportWeather.find(Filters.and(Filters.eq("airport", airport), Filters.gte("dt_1", timestamp))).first();
2
3 if (Math.abs(dt_local - timestamp) < 5500) {
```

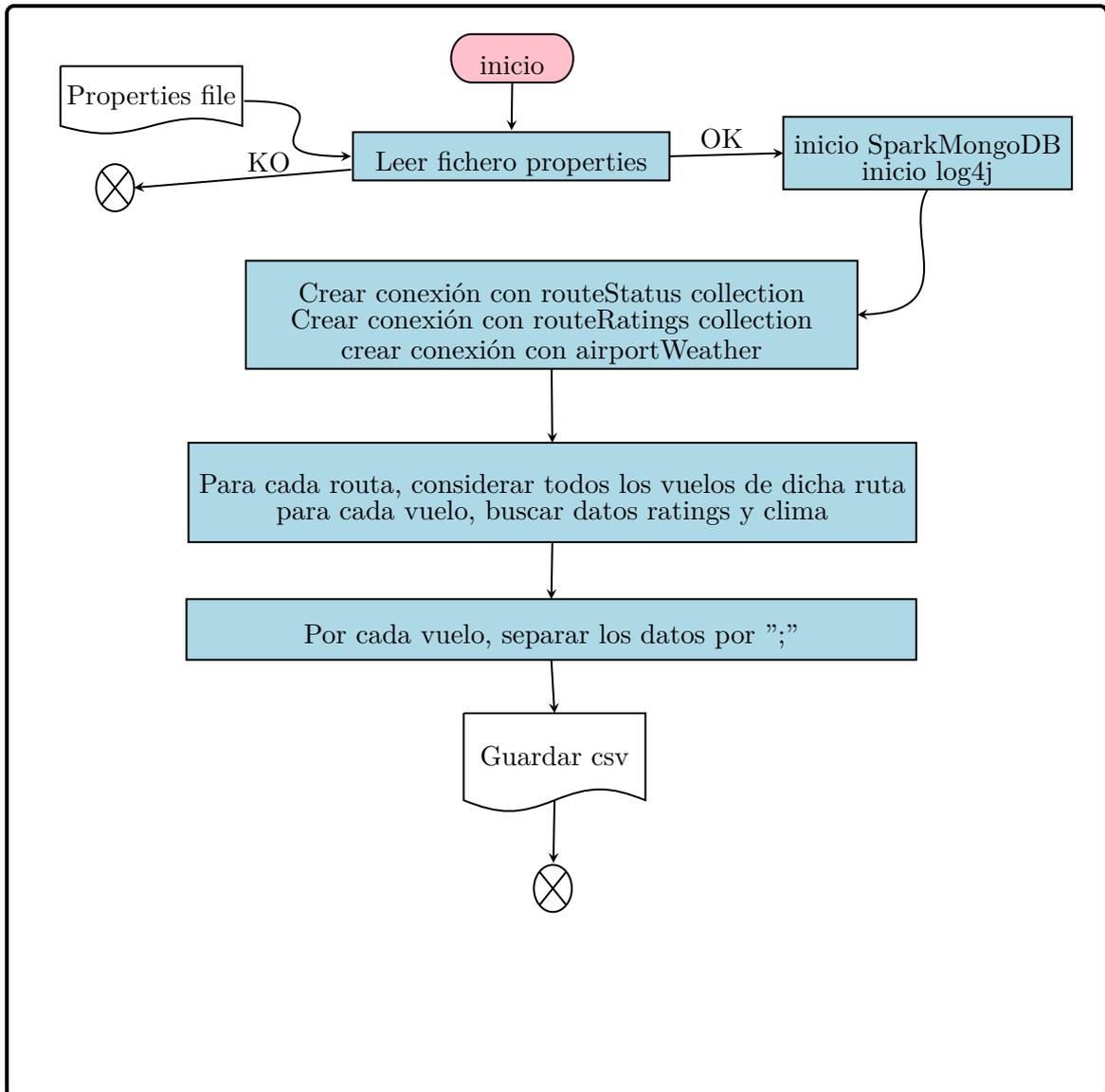
```

4     found = true;
5     out = weather_3h_i;
6     }

```

Además, se añade un índice a la colección airportWeather por el campo Aiports, dicho índice ha reducido el tiempo de acceso a la mitad aproximadamente. Un índice múltiple por airport y dt_1 ha sido probado donde el resultado no ha sido mejor que un índice simple.

3.1.5. Dataset.jar



Dicho programa nos sirve para crear un CSV con información relacionada con vuelos, y a partir de ahí entrenar los modelos, hacer análisis de datos, obtener gráficas entre otros propósitos.

Requisitos:

- Fichero de properties A
- Sesión MongoDB

Resultado:

Fichero csv.

La lógica del programa es:

Montar toda la colección de routeStatus en memoria. Por cada documento, nuclear sobre sus vuelos, para cada vuelo buscar datos del clima que le corresponden según el aeropuerto salida y la fecha y hora. Nos ayudamos de índices de MongoDB para agilizar dicha búsqueda en la colección de datos del clima. En cuanto a ratings, por cada vuelo buscamos sus estadísticas por id, ya que ésta está diseñada para acceder directamente a los datos del vuelo. Obteniendo así un CSV donde cada fila corresponde a un vuelo concreto con distintas features sobre el vuelo y que veremos en capítulo 4.

Testing

En un primer momento, sin volver a redefinir la clave de routeRatings ni crear índice para la colección airportWeather, el batch tardaba 41 minutos para formar el CSV con un total de 140 mil vuelos en la colección de routeStatus. Posteriormente, tras modificar la colección de routeRatings donde cada vuelo es un documento accesible por id bien escogida y no como antes, que cada vuelo era documento embebido en la ruta correspondiente, y tras añadir un índice por airport a airportWeather, el batch pasó a tardar únicamente 3.7 minutos para 190 mil vuelos en la colección routeStatus. Sin duda una mejora más que significativa que demuestra una vez más que en las bases de datos de documentos, el cómo se diseña la colección es clave en el performance.

Sin embargo, se podría mejorar aun más para ser escalable a medida que aumentan los datos, para explicar el cómo, primero vemos unos problemas con los que nos hemos cruzado al querer generar el CSV de forma distribuida y en paralelo.

- No podemos llamar a un estado de datos (una transformación RDD) desde otra, luego no podemos buscar información en otra colección estando en otra diferente. Esto por ejemplo limita el hecho de cargar con mongoSpark las tres colecciones y estando en JavaMongoRDD correspondiente a routeStatus, ir a buscar datos de clima y ratings a otras transformaciones.
- Si dentro de una transformación que gestiona los datos de una colección queremos buscar directamente en MongoDB y no otras transformaciones para solventar el problema anterior, necesitamos o bien serializar un objeto tipo JavaMongoRDD para que sea accesible desde otra transformación vía funciones definidas por usuario UDF (cosa que no está implementada en las librerías de Spark sql que ofrecen la posibilidad de serializar funciones UDF), o bien acceder directamente a la colección y por tanto se ha de crear conexiones dentro de la transformación RDD, lo que genera excesivas conexiones a mongo lo que provoca que no soporte y acabe fallando.

Por los problemas anteriores, y por ser un proceso batch que no se lanza frecuentemente sino únicamente al querer entrenar modelos, con una frecuencia mensual, hemos decidido que es suficiente una primera solución como la desarrollada. Pero dejamos unos razonamientos que sirvan a futuras evoluciones.

Los dos problemas anteriores nos hacen pensar que para usar Spark en distribuido, tener los datos en diferentes colecciones no es del todo efectivo, aunque en un principio parezca lógico y ordenado. Se debe crear una estructura de datos que unifique las tres fuentes de datos. Veamos un análisis de diferentes escenarios con el objetivo de unificar las tres colecciones:

Recordemos que la aplicación de búsqueda de vuelos, necesita acceso en tiempo real a datos del clima y de ratings para los vuelos que considere, luego estas colecciones se han de mantener con su estructura que se ha probado que es óptima, pues el acceso a ratings es por id y la colección es “estática”(no sufre muchas actualizaciones frecuentes ni crece mucho en tamaño) y el acceso a airportWeather es por índice que también ha mostrado un rendimiento bastante bueno. Con lo cual en definitiva, esa información se ha de mantener. Luego, para hacer escalable el programa que genera el CSV para entrenamiento y guardarlo en HDFS, necesitamos duplicar dicha información dentro de otra colección que contenga información de las tres colecciones.

Por otra parte nos viene bien duplicar la información del clima y que para tener el retraso de un vuelo, que es nuestra variable a predecir, nos tenemos que esperar un día después de salida del vuelo, y la versión gratuita de datos del clima, no da datos históricos para consultar en el momento de formar la colección, así que la consultaríamos de MongoDB ya guardada con antelación. Si usamos la versión de pago, no hay ningún problema, ya que en el momento de consultar routeStatus de forma distribuida con el programa routeStatusSpark, consultamos la API de ratings y la API de clima y generamos un documento con todo.

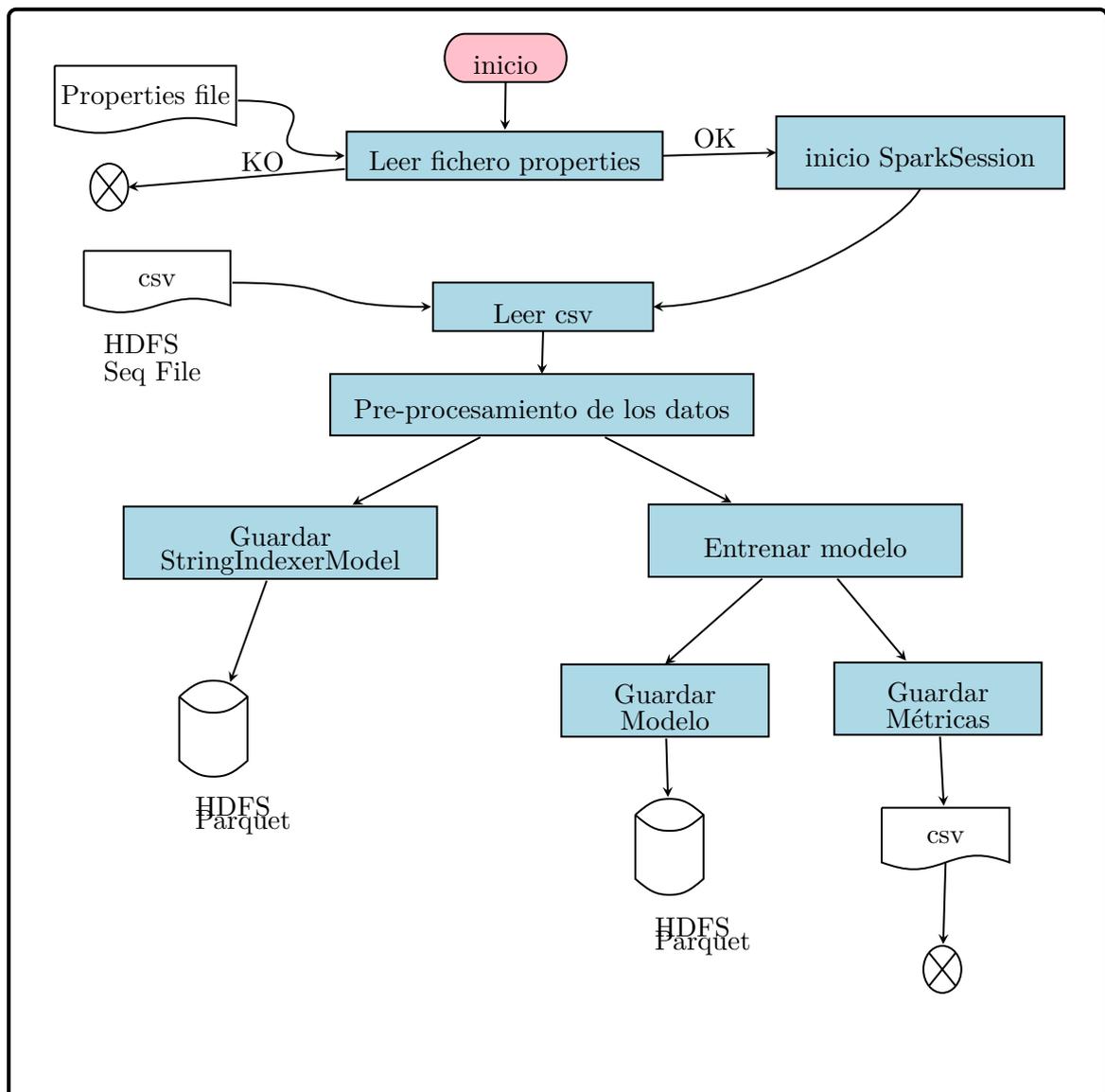
Pero para la versión que usamos en el prototipo (versiones API de evaluación), el programa RouteStatusSpark que baja datos de vuelos por rutas, debería incluir acceso a ratings API y entonces formatear el documento incluyendo a cada vuelo sus estadísticas. Ésto es posible sin ningún problema. Luego, al mismo tiempo consultar en mongo datos del clima, cosa que daría el mismo problema que antes, muchas conexiones a MongoDB que haría el programa inestable. La solución es filtrar los documentos por fecha, la que nos interese, guardarlos en fichero temporal y entonces, hacer lectura de fichero como si de una API se tratase, la desventaja es perder la performance de búsqueda que daría una base de datos como mongoDB.

Otro escenario, es usar una **arquitectura lambda**. El programa encargado de buscar vuelos para el cliente, consulta en tiempo real datos del clima y ratings de todos los vuelos que ha encontrado (la desventaja es que en tiempo real se añade latencia de red frente a latencia de consultar en bases propias), les aplica un tratamiento simple como unificarlos en un mismo fichero y guardarlo en Datalake. Posteriormente, un batch se encarga de coger todos los ficheros y tratarlos, pasar los tratados a nueva ubicación y suprime los ficheros tratados del Datalake. Un día mas tarde, se consulta por API flightStatus el estado de los vuelos del día anterior incluyendo el retraso y entonces guardar en MongoDB los documentos y borrar los ficheros del día anterior. Y en la speed Layer, llamar al modelo y predecir el retraso para ofrecer al usuarios los vuelos ordenados según sus expectativas.

Esta última solución o la de disponer de datos del clima en tiempo real pero históricos (versión OpenWeather de pago) serían buenas soluciones para hacer el programa dataset escalable.

Una arquitectura lambda combinada con sistema de ficheros sería lo ideal, pero es una implementación que requiere de tener el estudio de modelado de machine learning bloqueado hasta su finalización, cosa que por tiempo no nos hemos podido permitir. Y además, la API connections que nos ofrece vuelos futuros solo ofrece 500 conexiones al mes de forma gratuita, eso quiere decir que tendríamos pocos vuelos para entrenar los modelos⁵. Por estas razones, junto con las anteriores referidas a lógicas de RDDs, hemos decidido de desarrollar dataset.jar como aplicación temporal que nos permita no cruzarnos con problemas de mongoSpark, ni de tener que usar datos de pago de las APIs.

3.1.6. Model.jar



⁵Gran parte del trabajo con arquitectura Lambda, generación de ficheros en Datalake, está desarrollada, sin embargo, al tener límite en acceso datos por API connections a tan solo 500, se ha dejado de mantener ese desarrollo.

Model.jar es un programa que nos permite entrenar la mayoría de modelos de clasificación que tenemos disponibles en la librería MLib de Spark ⁶. Nos permite también generar tanto modelos de clasificación binaria como múltiple y también generar métricas que se almacenan en un CSV para posteriormente poder comparar modelos entre sí. Es un programa completo, con simplemente modificar el fichero de parámetros poder disponer de diferentes modelos que se almacenan de una manera ordenada para su posterior uso.

Requisitos:

- Fichero properties A
- csv

resultado:

Modelo entrenado junto a StringIndexerModel ambos guardados para su posterior uso en predicciones de retrasos.

La lógica del programa permite:

- **Entrenar diferentes modelos**, véase el fichero de properties A.

Testing

En cuanto al rendimiento, dicho programa es escalable y permite una ejecución sobre sistema distribuido HDFS. Los tiempos de ejecución dependen de los modelos escogido así como de los hiperparámetros que hayan usado a la hora de configurar los modelos en CrossValidation Model.

3.1.7. Sobre la arquitectura de datos

Notemos que para almacenar datos hemos usado MongoDB y para entrenar los modelos Spark MLib.

En cuanto a Data Management, recibimos los datos en formato Document, los documentos pueden diferir en estructura o tener atributos sin información. También, a priori el esquema no se puede definir pues estaríamos forzando la información que queremos tener y cómo la queremos tener cuando puede que documentos estén más ricos en información que otros, por lo que una estructura schemaless es sin duda más apropiada.

Unas alternativas técnicas podrían ser MongoDB, Elasticsearch o HBase.

ElasticSearch no sería una opción interesante para nuestro caso. En primer lugar, indexar todos los atributos de todos los documentos no nos aporta utilidad cuando el interés es acceder a documentos por su clave en la mayoría de casos y lo que nos podría provocar es una escritura lenta, y queremos dejar la posibilidad de escribir en tiempo real algún documento que recibamos por llamada a la API connections. Otra limitación es la imposibilidad de crear documentos embebidos, una estructura así generaría millones de documentos lo que podría

⁶<https://spark.apache.org/docs/2.3.0/ml-classification-regression.html>

provocar lentitud en el sistema.

Considerar HBase sobre HDFS, donde sólo tengamos que fijar las familias, por ejemplo tener tres familias `routeStatus`, `routeRatings` y `clima`. Dentro de cada familia tener diferentes `qualifiers`. Aquí, se debe tener toda la información estructurada por vuelo y no por ruta, para así la clave referirse al vuelo y tener la información de `ratings` y `clima` en las otras dos familias. El número de claves sería de millones, y la clave ha de tener cierta complejidad para permitir hacer búsquedas no solo por vuelos sino por rutas o aeropuertos. Muchos vuelos son rutinarios, con lo cual existirán con el tiempo y cada vez más, claves cercanas entre sí, lo que a HBase no le va bien por el balanceo. Además, no todos los mismos aeropuertos o rutas disponen del mismo flujo aéreo, con lo cual, esto tampoco favorece un balanceo en las regiones que se crean. Otra observación al respecto, es que usando HBase con gran cantidad de datos, se debe evitar `full scan table`, cuando aquí, para construir CSV de todos los vuelos, se necesita un `full scan`. Aunque HBase es una opción bastante efectiva en escalabilidad y particionamiento (vertical + horizontal) y en una versión de negocio podría ser usada si se encontrara la clave “ideal” y la estructura más apropiadas; no la descartaríamos como solución a ese problema.

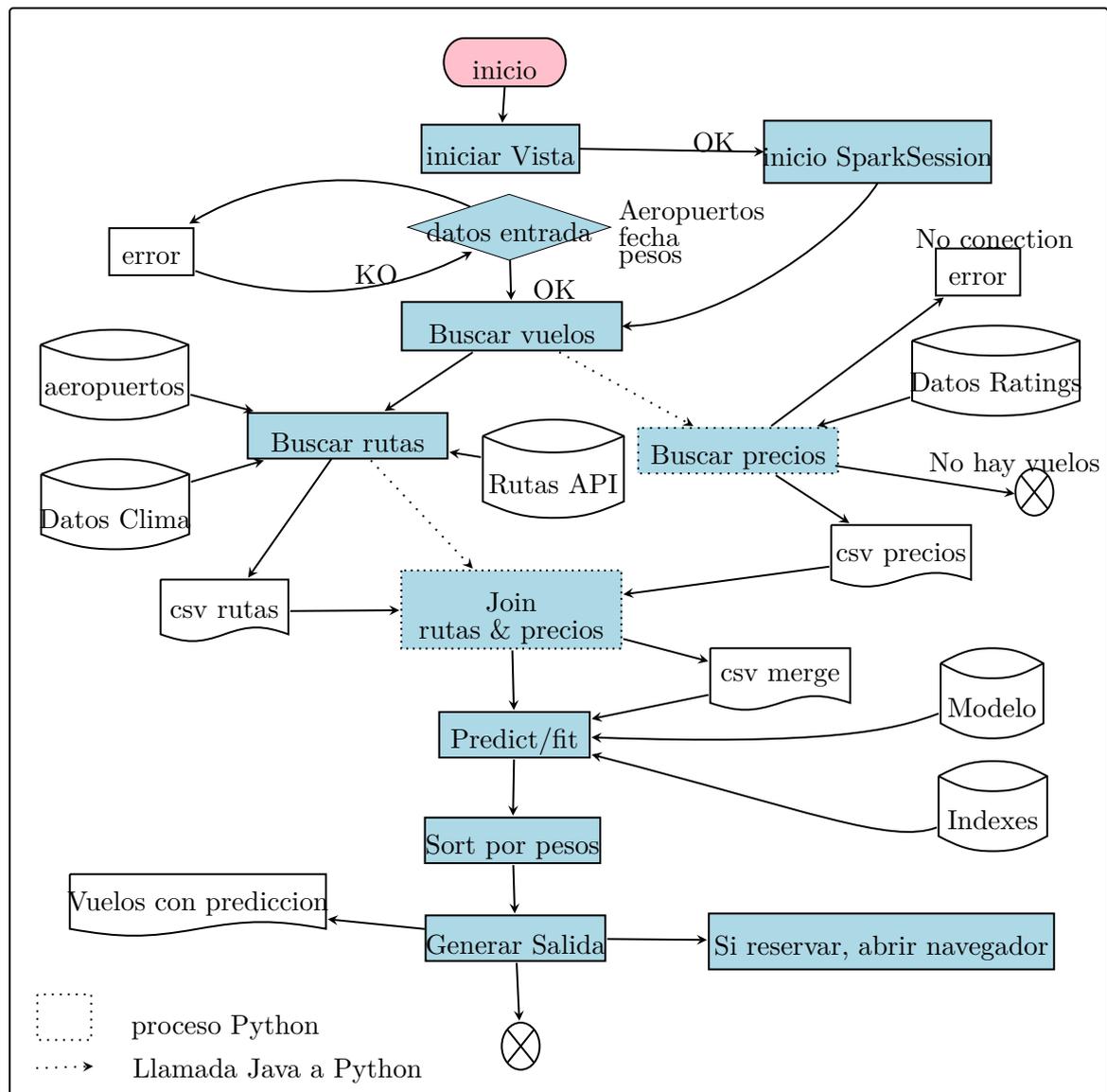
MongoDB, tiene algunas limitaciones: la cantidad de colecciones por base de datos, no optimiza un `acces plan` a la hora de usar índice, sólo acepta queries sobre la misma colección, no permite muchos documentos embebidos en otro documento, tiene un límite de documentos por colección ó el problema de `paddings` con los `updates`. Sin embargo, nosotros sólo necesitamos tres colecciones, los documentos embebidos no suponen ningún problema, son pocos en todas las colecciones: en `routeStatus` son los vuelos por ruta y día, en `routeRatings` no tenemos documentos embebidos y en el `airporWeather`, son siempre el mismo número de sub-documentos -40- (los correspondientes al clima de 5 días en intervalos de 3h). Los índices que usamos son claros, dado que el acceso que necesitamos optimizar es un acceso directo por un determinado atributo, no hay índices que elegir, mas que el único que hay. Los documentos una vez insertados, rara vez necesitan `update`, luego evitamos el problema de `paddings`. Con lo cual, una buena opción sería usar MongoDB, ya que para nuestros propósitos, no entramos en conflicto con sus puntos “débiles” antes mencionados.

En cuanto Spark MLlib, el objetivo es entrenar datos, que en realidad y si tuviéramos acceso ilimitado a éstos, serían muchísimos datos, dado que al año tienen lugar millones de vuelos. Y como el objetivo es hacer un prototipo pero teniendo muy presente la escalabilidad de nuestros programas y arquitecturas, el uso de librerías de Machine Learning en Distribuido y paralelo son necesarios. De ahí el uso de Spark MLlib.

3.2. Módulo búsqueda vuelos

El Módulo de Búsqueda de Vuelos hace referencia al subsistema de OnTime que se ocupa de servir la petición del usuario cuando desea buscar vuelos.

El diagrama siguiente describe las tareas efectuadas por el módulo.



3.2.1. Datalake

El Datalake es el repositorio en el que el sistema va dejando información necesaria para los diferentes procesos. Aunque para el prototipo no contenga toda la información posible, sí contiene un subconjunto de datos que representa la totalidad. Además, serviría en una arquitectura lambda para hacer del programa Dataset.jar 3.1.5 escalable.

Un ejemplo son los datos de las rutas, descargados de la API flightstats. En un primer momento, el sistema iba a conectarse diariamente a la API Connections para descargarse todas las rutas posibles en un proceso batch. Esto suponía que el tiempo de respuesta en el momento de buscar vuelo se vería reducido, puesto que además en ese momento sería posible asociar información como ratings o clima, y por tanto tener la información de los vuelos ya actualizada y las peticiones de usuario se verían reducidas en tiempo. Sin embargo, debido a no disponer de cuentas de pago para acceder volúmenes mayores de información, las rutas en el prototipo se descargan en la propia petición.

El Datalake, también se encarga de guardar las búsquedas ya realizadas con su predicción, para posteriormente testear la actitud del modelo en producción.

Fichero de aeropuertos

El módulo tiene en cuenta el universo de aeropuertos considerados para este prototipo, pues como se comentó en la sección módulo data analytics 3.1.1, reducimos el espacio de aeropuertos a 1.386 aeropuertos de los más de 16.000 existentes. Dicho fichero sirve también para el desplegable de la aplicación y cuenta con los aeropuertos ordenados alfabéticamente por nombre ciudad (Fichero B).

Rutas

Las rutas que consulta el módulo mediante la API flightStatus Connections, se guardan en el Datalake, posteriormente la aplicación permite consultar datos más frescos a la hora de hacer la petición el usuario, o tomar los existentes en el Datalake (esta última opción nos permite hacer tests sin gastar llamadas a la API, la versión final siempre toma datos actualizados en tiempo real).

Las variables respuesta en formato JSON que devuelve la API, pueden consultarse en `?`, pero fundamentalmente, la API Connections nos da vuelos entre dos aeropuertos incluyendo escalas en caso de haber, también incluye información sobre aerolíneas involucradas, aeropuertos y la equipación de las aeronaves.

Para más información consulte el anexo B.

3.2.2. Parámetros

Los parámetros necesarios para la aplicación, se encuentran en formato XML, fichero integrado a la aplicación que permite parametrizar diferentes aspectos.

Respecto del DataLake

- datalake-dir. Directorio donde se encuentra el Datalake.
 - flightstas-dir. Directorio donde se encuentran rutas descargadas de la API flightstats.
 - openflights-dir. Directorio donde se encuentran datos de la web openflights.
 - cfg-dir. Directorio que contiene fichero airports_final.txt, el universo de aeropuertos del prototipo.
-

- airports-univers-file. Nombre del fichero que define el universo de aeropuertos del prototipo.

Respecto de la API flighttas

- appId. Id de aplicación para ir a buscar datos a la API.
- appKey. Key para ir a buscar datos a la API.

Parámetros relacionados con api Connections de flightstats <https://developer.flightstats.com/api-docs/connections/v2>:

- maxConnections. Máximo número de escalas intermedias.
- numHours. Horas hacia atrás en las que mirar vuelos (firstflightin).
- maxResults. Máximo número de resultados.
- includeSurface. Si se incluyen conexiones de transporte de superficie.
- payloadType. Conexiones de pasajeros, carga o ambas.
- includeCodeshares. Inclusión de las gestiones de aerolíneas alternativas.
- includeMultipleCarriers. Inclusión de múltiples aerolíneas.

Tipo API Connection firstflightIN. Vuelos **que llegan antes** de la hora indicada. Mediante este tipo de llamadas a la API conseguimos información de vuelos que llegan al día siguiente del vuelo (pero no que salen ese día), y que forman parte del cálculo de las rutas:

- firstflightin-dir. Directorio connections-firstflightin.
- firstflightin-url. URL template connections-firstflightin.
- firstflightin-hour. Momento a partir del cual mirar hacia atrás.
- firstflightin-min. Momento a partir del cual mirar hacia atrás.

Tipo API Connection firstflightOUT. Vuelos que **se van después** de la hora indicada. Mediante este tipo de llamadas a la API conseguimos información de vuelos que salen (y llegan) el mismo día del vuelo, y que forman parte del cálculo de las rutas:

- firstflightout-dir. Directorio connections-firstflightout.
 - firstflightout-url. URL template connections-firstflightout.
 - firstflightout-hour. Momento a partir del cuál mirar hacia adelante.
 - firstflightout-min. Momento a partir del cuál mirar hacia adelante.
-

Respecto de la Interfaz con Python

- python-dir. Directorio donde se encuentran los scripts de python y los ficheros de intercambio con el módulo de rutas
- merge-csv-dir. Directorio donde quedan los resultados de las rutas posibles para servir al usuario, incluyendo Duración y Retraso previsto.

Respecto del Modelo de predicción

- model-dir. Directorio donde obtener modelo entrenado.
- model-index-dir. Directorio donde obtener índices del modelo, correspondientes a la indexación de las variables.
- model-name. Nombre/Token del modelo usado para la predicción. Todos los disponibles se hallan en el fichero A.

Otros

- csv-separator. Separador de ficheros csv.
- timeZone. Offset de la zona horaria donde se ejecuta la aplicación.

3.2.3. Algoritmo

La búsqueda de vuelos sigue el diagrama indicado anteriormente. A continuación se indican las características del algoritmo de Búsqueda de vuelos, así como los requisitos y el resultado que genera.

Características

En primer lugar, tras inicializar la aplicación (interfaz, acceso a Spark, acceso a MongoDB, etc.), se muestra al usuario el prototipo de búsqueda de vuelos con retardo. El usuario debe escoger, origen del vuelo, destino, y una fecha, así como asignar pesos a las tres características que servirán para efectuar un ordenamiento ponderado de las rutas encontradas.

Cuando el usuario clicca el botón de Buscar Vuelo, se desencadenan una serie de acciones que se comentan a continuación.

Tras recoger los parámetros introducidos por el usuario, el sistema efectúa la llamada al módulo de Python de búsqueda de precios. Este módulo utiliza el origen, destino y fecha para efectuar un scraping a la web kayak <https://www.kayak.es/> que obtenga precios para ese vuelo demandado. Este módulo deja sus resultados en fichero CSV en un directorio configurado a tal efecto y con una estructura concreta (ver fichero de rutas generado, a continuación).

Este proceso de obtención de precios también se usa para completar cierta información más concreta, como la duración de los vuelos, la url para comprar ese vuelo concreto, o información de codeshares que la API devolvería en caso de disponer de cuentas de acceso total, lo

cuál no es el caso.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1		A.Salida	A.Llegada	Duracion	Paradas	Vuelos alter precio	Escala	hSalida_v1	hLlegada_v1	hSalida_v2	hLlegada_v2	Enlaces	Code_v1	flightnumbe	Code_v2	flightnumber_v2		
2	0	AUH	AMS	13h 30m	1escala	0	828 IST	1.35	5.25	10.35	13.05	https://www	TK	869 TK		1957		
3	1	AUH	AMS	17h 55m	1escala	0	946 IST	1.35	5.25	15.05	17.30	https://www	TK	869 TK		1953		
4	2	AUH	AMS	10h 15m	1escala	0	1064 IST	1.35	5.25	7.15	9.50	https://www	TK	869 TK		1951		
5	3	AUH	AMS	10h 15m	1escala	0	5000 IST	1.35	5.25	7.15	9.50	https://www	EY	4151 TK		1951		
6	4	AUH	AMS	17h 50m	None	0	747 LHR-LCY	1.45	6.25	15.30	17.35	https://www	BA	72 BA		8455		
7	5	AUH	AMS	17h 15m	1escala	0	757 LHR	1.45	6.25	14.40	17.00	https://www	BA	72 BA		432		
8	6	AUH	AMS	14h 20m	1escala	0	774 LHR	1.45	6.25	11.45	14.05	https://www	BA	72 BA		434		
9	7	AUH	AMS	13h 30m	None	0	791 LHR-LCY	1.45	6.25	17.05	19.15	https://www	BA	72 BA		8459		
10	8	AUH	AMS	18h 50m	1escala	0	813 LHR	1.45	6.25	16.15	18.35	https://www	BA	72 BA		440		
11	9	AUH	AMS	20h 35m	1escala	0	813 LHR	1.45	6.25	18.00	20.20	https://www	BA	72 BA		442		
12	10	AUH	AMS	13h 10m	1escala	0	857 LHR	1.45	6.25	10.35	12.55	https://www	BA	72 BA		436		
13	11	AUH	AMS	15h 10m	1escala	0	857 LHR	1.45	6.25	12.35	14.55	https://www	BA	72 BA		438		
14	12	AUH	AMS	13h 20m	None	0	916 LHR-LGW	1.45	6.25	10.40	13.05	https://www	BA	72 BA		2760		
15	13	AUH	AMS	10h 55m	1escala	0	917 LHR	1.45	6.25	8.20	10.40	https://www	BA	72 BA		430		
16	14	AUH	AMS	16h 40m	None	0	999 LHR-LGW	1.45	6.25	16.10	18.25	https://www	BA	72 BA		2762		
17	15	AUH	AMS	20h 40m	None	0	999 LHR-LGW	1.45	6.25	18.10	20.25	https://www	BA	72 BA		2764		
18	16	AUH	AMS	15h 25m	None	0	1030 LHR-LCY	1.45	6.25	13.05	15.10	https://www	BA	72 BA		8453		
19	17	AUH	AMS	15h 45m	1escala	0	1550 BCN	2.00	7.20	13.20	15.45	https://www	EY	49 EY		7423		
20	18	AUH	AMS	14h 30m	1escala	0	1574 BCN	2.00	7.20	12.05	14.30	https://www	EY	49 VY		8302		

Figura 3.2: Muestra csv precios

De manera paralela, el sistema efectúa dos llamadas a la API flightstats para obtener datos de rutas del vuelo demandado en formato CSV:

- Datos de rutas que salen del aeropuerto origen para el día introducido por el usuario ('DSalida') y que llegan el mismo día.

Esta llamada es de tipo "FirstFlightOut" en la nomenclatura de la API Flightstats, y significa que se quieren considerar los vuelos **que salen después de la hora indicada** durante un tiempo determinado. En este caso (vía la parametrización vista anteriormente), se buscan vuelos en "DSalida" que salen después de las 0h mirando hacia adelante 24h para cubrir todo el día.

- Datos de rutas que salen en DSalida, pero que pueden llegar en (DSalida+1).

Para ello se efectúa una petición a la API de tipo 'FirstFlightIn'. Esto significa que se buscan vuelos para el día siguiente a la petición del usuario **que llegan antes de la hora indicada**, durante un tiempo determinado. En este caso, se obtienen los vuelos para el día (DSalida+1) desde las 23h y 59min, mirando hacia atrás 24h (desde las 0h de ese día).

Con estas dos consultas a la API flightstats, se pueden obtener todas las rutas que se ven implicadas en la consulta efectuada por el usuario. Esas rutas obtenidas, se transforman para obtener un csv **con una estructura similar**, que se genera en el módulo de Python de consulta de precios (GetPrices), dado que a posteriori se efectuará una fusión entre ellos.

La información que el csv de rutas contiene, básicamente, es la siguiente:

- Información del usuario en su petición.
- Si el vuelo es directo o no.
- Información de la primera escala (o información del vuelo, si es directo y no hay escalas). Esta información incluye:

- Aeropuertos origen y destino.
 - Aerolínea o carrier.
 - Número de vuelo.
 - Fecha y hora de salida.
 - Fecha y hora de llegada.
 - Codeshares (gestión del mismo vuelo por parte de otra aerolínea)
- Información de la segunda escala (si existe). Similar a lo anterior.
 - Datos adicionales⁷, interesantes de cara a la predicción del retraso, como la información del clima de los aeropuertos origen en el momento (no exacto⁸) de partida.

departi	arri	departu	Da w	Pre v	Ret	wDur	id	Con	directo	depart	FSalida_v1	hSalida	carri	flightN	arri	FLlegada_v1	hLlega	codesh	temp	humid	pressu	weather	windSp	windD	departi	FSalida_v2	hSalida	carri	flightN	arri	FLlegada_v2	hLlega	v2	codesh	temp	
BCN	DME	20191025	1	3	2	BCNDME_20191025_1	true	BCN	25/10/2019	14:45	U6	846	DME	25/10/2019	20:10	[]	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null
BCN	DME	20191025	1	3	2	BCNDME_20191025_2	true	BCN	25/10/2019	15:05	S7	892	DME	25/10/2019	20:30	[]	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null
BCN	DME	20191025	1	3	2	BCNDME_20191025_3	false	BCN	25/10/2019	1:00	9U	598	KIV	25/10/2019	5:20	[]	null	null	null	null	null	null	null	null	null	KIV	25/10/2019	7:10	S7	4538	DME	25/10/2019	8:50	[[9U-null]]	null	
BCN	DME	20191025	1	3	2	BCNDME_20191025_4	false	BCN	25/10/2019	1:00	9U	598	KIV	25/10/2019	5:20	[]	null	null	null	null	null	null	null	null	null	KIV	25/10/2019	7:10	9U	173	DME	25/10/2019	8:50	[[S7-4538]]	null	
BCN	DME	20191025	1	3	2	BCNDME_20191025_5	false	BCN	25/10/2019	1:00	9U	598	KIV	25/10/2019	5:20	[]	null	null	null	null	null	null	null	null	null	KIV	25/10/2019	13:50	9U	171	DME	25/10/2019	15:30	[]	null	
BCN	DME	20191025	1	3	2	BCNDME_20191025_6	false	BCN	25/10/2019	1:00	9U	598	KIV	25/10/2019	5:20	[]	null	null	null	null	null	null	null	null	null	KIV	25/10/2019	13:50	S7	4538	DME	25/10/2019	15:30	[[9U-null]]	null	
BCN	DME	20191025	1	3	2	BCNDME_20191025_7	false	BCN	25/10/2019	1:00	9U	598	KIV	25/10/2019	5:20	[]	null	null	null	null	null	null	null	null	null	KIV	25/10/2019	13:50	9U	173	DME	25/10/2019	15:30	[[S7-4538]]	null	
BCN	DME	20191025	1	3	2	BCNDME_20191025_8	false	BCN	25/10/2019	6:40	U2	1404	GVA	25/10/2019	8:15	[]	null	null	null	null	null	null	null	null	GVA	25/10/2019	10:15	LX	1336	DME	25/10/2019	15:00	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_9	false	BCN	25/10/2019	7:00	IB	5056	MUC	25/10/2019	9:10	[[VY-null]]	null	null	null	null	null	null	null	null	MUC	25/10/2019	11:05	LH	2528	DME	25/10/2019	15:10	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_10	false	BCN	25/10/2019	7:00	VY	null	MUC	25/10/2019	9:10	[[IB-5056]]	null	null	null	null	null	null	null	null	MUC	25/10/2019	11:05	LH	2528	DME	25/10/2019	15:10	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_11	false	BCN	25/10/2019	6:05	FR	6341	FCO	25/10/2019	8:00	[]	null	null	null	null	null	null	null	null	FCO	25/10/2019	15:45	S7	626	DME	25/10/2019	20:30	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_12	false	BCN	25/10/2019	7:55	LH	1817	MUC	25/10/2019	9:55	[]	null	null	null	null	null	null	null	null	MUC	25/10/2019	11:05	LH	2528	DME	25/10/2019	15:10	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_13	false	BCN	25/10/2019	7:00	IB	5056	MUC	25/10/2019	9:10	[[VY-null]]	null	null	null	null	null	null	null	null	MUC	25/10/2019	13:50	S7	796	DME	25/10/2019	17:55	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_14	false	BCN	25/10/2019	7:00	VY	null	MUC	25/10/2019	9:10	[[IB-5056]]	null	null	null	null	null	null	null	null	MUC	25/10/2019	13:50	S7	796	DME	25/10/2019	17:55	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_15	false	BCN	25/10/2019	6:50	LH	1139	FRA	25/10/2019	9:00	[]	null	null	null	null	null	null	null	null	FRA	25/10/2019	13:05	LH	1448	DME	25/10/2019	17:15	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_16	false	BCN	25/10/2019	6:50	WY	5539	FRA	25/10/2019	9:00	[[LH-null]]	null	null	null	null	null	null	null	null	FRA	25/10/2019	13:05	LH	1448	DME	25/10/2019	17:15	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_17	false	BCN	25/10/2019	6:50	LH	null	FRA	25/10/2019	9:00	[[WY-5539]]	null	null	null	null	null	null	null	null	FRA	25/10/2019	13:05	LH	1448	DME	25/10/2019	17:15	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_18	false	BCN	25/10/2019	6:25	AZ	75	FCO	25/10/2019	8:10	[]	null	null	null	null	null	null	null	null	FCO	25/10/2019	15:45	S7	626	DME	25/10/2019	20:30	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_19	false	BCN	25/10/2019	6:25	UK	3124	FCO	25/10/2019	8:10	[[AZ-null]]	null	null	null	null	null	null	null	null	FCO	25/10/2019	15:45	S7	626	DME	25/10/2019	20:30	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_20	false	BCN	25/10/2019	6:25	AZ	null	FCO	25/10/2019	8:10	[[UK-3124]]	null	null	null	null	null	null	null	null	FCO	25/10/2019	15:45	S7	626	DME	25/10/2019	20:30	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_21	false	BCN	25/10/2019	6:25	HU	8664	FCO	25/10/2019	8:10	[[AZ-null]]	null	null	null	null	null	null	null	null	FCO	25/10/2019	15:45	S7	626	DME	25/10/2019	20:30	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_22	false	BCN	25/10/2019	6:25	AZ	null	FCO	25/10/2019	8:10	[[HU-8664]]	null	null	null	null	null	null	null	null	FCO	25/10/2019	15:45	S7	626	DME	25/10/2019	20:30	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_23	false	BCN	25/10/2019	8:00	LH	1137	FRA	25/10/2019	10:10	[]	null	null	null	null	null	null	null	null	FRA	25/10/2019	13:05	LH	1448	DME	25/10/2019	17:15	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_24	false	BCN	25/10/2019	7:55	LH	1817	MUC	25/10/2019	9:55	[]	null	null	null	null	null	null	null	null	MUC	25/10/2019	13:50	S7	796	DME	25/10/2019	17:55	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_25	false	BCN	25/10/2019	7:20	BA	8093	FCO	25/10/2019	9:10	[[VY-null]]	null	null	null	null	null	null	null	null	FCO	25/10/2019	15:45	S7	626	DME	25/10/2019	20:30	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_26	false	BCN	25/10/2019	7:20	VY	null	FCO	25/10/2019	9:10	[[BA-8093]]	null	null	null	null	null	null	null	null	FCO	25/10/2019	15:45	S7	626	DME	25/10/2019	20:30	[]	null		
BCN	DME	20191025	1	3	2	BCNDME_20191025_27	false	BCN	25/10/2019	7:20	IB	5512	NCE	25/10/2019	8:40	[[VY-null]]	null	null	null	null	null	null	null	null	NCE	25/10/2019	15:10	S7	656	DME	25/10/2019	20:05	[]	null		

Figura 3.3: Muestra csv routes

En la imagen del csv de rutas podemos observar algunas características:

- Las columnas **xx_v1** hacen referencia al “vuelo 1” o primer vuelo, y **xx_v2** hacen referencia al “vuelo 2” o segundo vuelo en la ruta. Si hay escala tendremos los datos **xx_v2** alimentados, en caso de que el vuelo sea directo, **xx_v2** contiene datos nulos. En la imagen 3.3, la parte sombreada hace referencia al vuelo 1, sucesivamente tenemos los datos del vuelo 2.
- Las primeras seis columnas hacen referencia a los datos que el usuario ha introducido. Los llamados **datos generales**.
- la columna **directo** indica si el vuelo es directo y por ende **xx_v2** estarán a null, o tiene escala.
- **idConexion** es un atributo que agrupa líneas que han sido extraídas de la misma conexión en la respuesta de la API. Mediante esta agrupación el sistema da información

⁷Datos de Ratings. Por una cuestión de optimización, estos datos se obtienen finalmente en el módulo de Python, JoinFile.

⁸Los datos del clima son los correspondientes al aeropuerto salida pero no en la hora exacta del vuelo, si no en un intervalo de $\pm 1.5h$.

adicional de los codeshares existentes para la misma ruta (ver columna `codeshares_v1` y `codeshares_v2`). En la figura 3.3 se muestran coloreados algunos grupos de rutas para una conexión.

- **Codeshares_v1/v2.** Columnas que indican qué alternativas hay en la escala correspondiente para un `numVuelo` y `carrier`. Es una lista de pares (`carrier`, `numVuelo`) con esas posibilidades. En ocasiones la API no proporciona alguno de esos datos en la versión gratuita que el prototipo usa. En estos casos, el proceso de Python `GetPrices` es el que setea este dato efectuando scraping.
- `temp_v1`, `humidity_v1`, `pressure_v1`, `weather_v1`, `windSpeed_v1`, `windDeg_v1` y sus correspondientes `xx_v2` son datos relativos al clima del aeropuerto (localidad más cercana) alrededor de la hora de salida del aeropuerto origen. En este caso no se dispone de datos de clima y por ello aparecen valores nulos en la figura 3.3.
- El resto de columnas hacen referencia a datos básicos de un vuelo, fecha salida, hora salida, compañía, número vuelo.

Una vez que se han generado los dos CSV con una estructura similar (el de las rutas posibles para la petición del usuario, y el CSV de precios de esas rutas via scraping), se debe fusionar esa información para obtener una estructura única en un tercer CSV.

Esta tarea la efectúa un segundo módulo de Python (`JoinFile`). Este módulo genera una estructura única que contiene las rutas de las que se han conseguido precios. No todas las rutas tienen por qué tener un precio definido en la web del proveedor en el momento en que se efectúa el scraping bien sea porque ya han agotado los billetes, bien Kayak no tiene acuerdo con todas las aerolíneas y proveedores de vuelos, por lo que al final no aparecerán en los resultados finales.

Asimismo se añaden datos de ratings 3.1.3, que igualmente `JoinFile` es el módulo que se encarga de incorporar estos al CSV.

En definitiva, ese segundo proceso Python genera un CSV “csv merge”⁹ que está preparado para ser completado con la información de la predicción de los retrasos de esas rutas.

En este fichero CSV, siguen existiendo dos grupos de columnas principales, los atributos relativos al primer vuelo, o al segundo.

Algunas características destacables del CSV son las siguientes:

- Se han añadido al CSV datos de ratings del vuelo, a saber: `ontime_v1`, `late15_v1`, `late30_v1`, `late45_v1`, `delayMean_v1`, `delayStandardDeviation_v1`, `delayMin_v1`, `delayMax_v1`, `allOntimeCumulative_v1`, `allOntimeStars_v1`, `allDelayCumulative_v1`, `allDelayStars_v1`¹⁰

Estos atributos como se dijo anteriormente, se añaden por el módulo `JoinFile` de Python por cuestiones de optimización, por una parte se evita crear múltiples conexiones con MongoDB, cosa que surge al usar Spark y Mongo, problema ya detectado y explicado en sección 3.1.5. Por otra parte, también por el hecho de tener codeshares sin `flightNumber` en la versión gratuita, muchos vuelos en el CSV rutas aparecen sin `flightNumber`. El

⁹No se incluye en la memoria ejemplo de este CSV por su tamaño, pero como dijimos fusiona datos de CSV precios 3.2 y CSV rutas 3.3

¹⁰Más información sobre ratings en 3.1.3.

módulo de Python GetPrices se encarga de completar dicha información y por tanto en JoinFile, sólo añadiremos los datos de ratings a aquellos vuelos que ya tienen flight-Number y precio. Optimizamos por tanto el número de búsquedas a MongoDB, y sólo buscamos una vez como mucho por idConnection.

- Al final del CSV vemos las columnas que había conseguido previamente el proceso GetPrices, incluyendo la URL para la compra y el precio del vuelo, entre otras.

Finalmente, se aborda la parte de la predicción del retraso de las rutas anteriores, para añadir esa información al dataset y obtener el resultado final.

Se dispone de un modelo entrenado para diferentes algoritmos (véase la sección 3.1.6), por lo que podría aplicarse uno u otro en función de la parametrización.

Así, se efectúa el “fit” de las rutas resultantes para obtener su predicción de retraso y se completa el dataset. El dataset resultante, se ordena para que cumpla las expectativas del usuario en relación a la asignación de pesos que indicó en la petición (atributos precio, duración y retraso). En función del orden de esos pesos (a qué da más importancia), visualizará en primer lugar unas rutas u otras.

La columna interna de ordenación se construye de la siguiente manera:

Sea S_i el score de la ruta i , entonces:

$$S_i = \frac{\omega_1 \tilde{X}_i + \omega_2 \tilde{Y}_i + \omega_3 \tilde{Z}_i}{1 + \omega_1 + \omega_2 + \omega_3}$$

donde, \tilde{X}_i corresponde al valor del precio para la ruta i normalizado, \tilde{Y}_i corresponde al valor de la duración para la ruta i normalizado y finalmente \tilde{Z}_i corresponde a una estimación total simplificada del retraso para la ruta i también normalizada.

El 1 sumando en el denominador, sirve para evitar divisiones por cero. La normalización nos es útil para tratar el orden sin errores de magnitudes diferentes y poder efectuar la suma de diferentes unidades.

Como último paso, el dataset resultante se inyecta en la tabla del interfaz de salida.

El usuario puede visualizar finalmente las rutas para el vuelo que demanda. Aparte de los datos de las propias rutas, mencionados anteriormente (datos de escalas, etc.), el usuario puede observar al final de la tabla los siguientes atributos:

- Href: URL que le llevará con un clic a la compra del vuelo que sigue esa ruta.
- Duración: la duración de la ruta.
- Retraso: la predicción de si esa ruta se va a retrasar.

En este punto, ya estamos preparados para buscar y reservar vuelos!!

Directo	Origen	Fecha1a	Hora1a	Aerolínea	Vuelo	Destino	Fecha1b	Hora1b	Origen2	Fecha2a	Hora2b	Aerolínea2	Vuelo2	Destino2	Fecha2b	Hor
true	BCN	2019-10-25	14:45	U6	846	DME	2019-10-25	20:10					nan			
false	BCN	2019-10-25	07:55	LH	1817	MUC	2019-10-25	09:55	MUC	2019-10-25	11:05	LH	2528	DME	2019-10-25	15:11
false	BCN	2019-10-25	13:50	LH	1131	FRA	2019-10-25	16:00	FRA	2019-10-25	17:10	LH	1450	DME	2019-10-25	21:2
false	BCN	2019-10-25	15:45	LH	1813	MUC	2019-10-25	17:45	MUC	2019-10-25	20:00	LH	2530	DME	2019-10-26	00:0
false	BCN	2019-10-25	18:30	VY	8102	ATH	2019-10-25	22:25	ATH	2019-10-25	23:30	A3	882	DME	2019-10-26	02:5
false	BCN	2019-10-25	12:45	LH	1127	FRA	2019-10-25	14:55	FRA	2019-10-25	17:10	LH	1450	DME	2019-10-25	21:2
false	BCN	2019-10-25	08:00	LH	1137	FRA	2019-10-25	10:10	FRA	2019-10-25	13:05	LH	1448	DME	2019-10-25	17:1
false	BCN	2019-10-25	06:40	U2	1404	GVA	2019-10-25	08:15	GVA	2019-10-25	10:15	LX	1336	DME	2019-10-25	15:0
false	BCN	2019-10-25	06:50	LH	1139	FRA	2019-10-25	09:00	FRA	2019-10-25	13:05	LH	1448	DME	2019-10-25	17:1

Figura 3.4: ontime Interfaz vuelos de BCN Barcelona a DME Moscow para el día 25-10-2019

Se observa que mostramos los datos necesarios para el usuario y no todos los datos del CSV merge, ni del dataset salida del “fit”. El usuario dispone de toda la información que necesita conocer a cerca de su viaje, en la columna **HREF** dispone del link para hacer la compra, bastaría tener un navegador definido como predeterminado en su sistema, para que, mediante un clic acceder a la compra totalmente segura de su billete.

Cabe destacar que la columna **Delayed** la usamos para informar del retraso. Aquí una simplificación temporal se ha hecho y es la siguiente:

- **0.0**, label que indica que ningún vuelo de nuestra ruta se espera que sufra ningún retraso.
- **15.0**, label que indica que uno de nuestros vuelos, se espera que sufra un retraso superior a 15 minutos.
- **30.0**, label que indica que ambos vuelos, en una ruta con escala, se espera que sufrirán retrasos de más de 15 minutos cada uno.

La simplificación reside en que, para **15.0** no precisamos en que escala es, a nivel interno de la aplicación esta información es conocida. Pero la simplificación elemental es en el caso de **30.0**, retraso en ambos vuelos que traducimos en retraso general cuando no lo es, pues puede darse el caso en que, aunque retrasados en el primer vuelo, llegamos con tiempo a tomar el segundo vuelo, por tanto ese retraso se traduce en menor tiempo de espera en la escala que en retraso general como esta ahora desarrollado en la aplicación.

La evolución del software debe tener en cuenta los diferentes horarios, estimar los retrasos y calcular tiempos de llegada primer vuelo con retraso estimado añadido, ver si es inferior al horario salida segundo aeropuerto con retraso sumado y entonces operar la espera en escala y ver si de verdad es superior a la escala sin retrasos, en cuyo caso habrá un retraso en la

ruta general, en otro caso, sólo contaría el retraso del primer vuelo.

Nótese que el ordenamiento de vuelos es según unos pesos, por lo cual no hay filtros, usted tendrá los vuelos ordenados según sus intereses en primer orden. Veamos algunos ejemplos para comprender este tipo de ordenación:

ontime launcher

Aeropuerto de salida: Barcelona (BCN) | Aeropuerto de llegada: Hong Kong (HKG) | Fecha del vuelo: 25/10/2019

Precio: 0 | Duración: 0 | Retraso (min): 100 | **Buscar vuelo**

Vuelos

Directo	Origen	Fecha1a	Hora1a	Aero...	Vuelo	Destino	Fecha1b	Hora1b	Origen2	Fecha2a	Hora2a	Aer...	Vuelo2	Des...	Fecha2b	Hora2b	Precio	Duración	HREF	Delayed
false	BCN	2019-10-25	00:05	SU	2513	SVO	2019-10-25	05:35	SVO	2019-10-25	19:05	SU	212	HKG	2019-10-26	09:55	359	27h 50m	https://w...	0.0
false	BCN	2019-10-25	06:50	LH	1139	FRA	2019-10-25	09:00	FRA	2019-10-25	13:45	CX	288	HKG	2019-10-26	06:50	636	18h 00m	https://w...	0.0
false	BCN	2019-10-25	07:10	BA	477	LHR	2019-10-25	08:35	LHR	2019-10-25	12:20	CX	252	HKG	2019-10-26	07:05	749	17h 55m	https://w...	0.0
false	BCN	2019-10-25	08:00	LH	1137	FRA	2019-10-25	10:10	FRA	2019-10-25	13:45	CX	288	HKG	2019-10-26	06:50	636	16h 50m	https://w...	0.0
false	BCN	2019-10-25	12:05	SQ	387	SIN	2019-10-26	07:00	SIN	2019-10-26	08:40	SQ	860	HKG	2019-10-26	12:40	817	18h 35m	https://w...	0.0
false	BCN	2019-10-25	12:05	SQ	387	SIN	2019-10-26	07:00	SIN	2019-10-26	08:00	CX	710	HKG	2019-10-26	12:05	2647	18h 00m	https://w...	0.0
false	BCN	2019-10-25	11:05	AY	1654	HEL	2019-10-25	16:00	HEL	2019-10-25	16:40	AY	101	HKG	2019-10-26	07:25	951	14h 20m	https://w...	15.0
false	BCN	2019-10-25	12:00	SU	2639	SVO	2019-10-25	17:40	SVO	2019-10-25	19:05	SU	212	HKG	2019-10-26	09:55	359	15h 55m	https://w...	15.0
false	BCN	2019-10-25	11:05	KL	1666	AMS	2019-10-25	13:35	AMS	2019-10-25	17:15	KL	887	HKG	2019-10-26	10:20	771	17h 15m	https://w...	30.0

Figura 3.5: ontime Interfaz vuelos de BCN Barcelona a HKG Hong Kong para el día 25-10-2019 y pesos Precio = 0, Duración = 0, Retraso = 100

ontime launcher

Aeropuerto de salida: Barcelona (BCN) | Aeropuerto de llegada: Hong Kong (HKG) | Fecha del vuelo: 25/10/2019

Precio: 100 | Duración: 0 | Retraso (min): 10 | **Buscar vuelo**

Vuelos

Directo	Origen	Fecha1a	Hora1a	Aero...	Vuelo	Destino	Fecha1b	Hora1b	Origen2	Fecha2a	Hora2a	Aer...	Vuelo2	Des...	Fecha2b	Hora2b	Precio	Duración	HREF	Delayed
false	BCN	2019-10-25	00:05	SU	2513	SVO	2019-10-25	05:35	SVO	2019-10-25	19:05	SU	212	HKG	2019-10-26	09:55	359	27h 50m	https://w...	0.0
false	BCN	2019-10-25	12:00	SU	2639	SVO	2019-10-25	17:40	SVO	2019-10-25	19:05	SU	212	HKG	2019-10-26	09:55	359	15h 55m	https://w...	15.0
false	BCN	2019-10-25	15:15	AF	1649	CDG	2019-10-25	17:10	CDG	2019-10-25	23:35	AF	188	HKG	2019-10-26	17:35	565	20h 20m	https://w...	30.0
false	BCN	2019-10-25	06:50	LH	1139	FRA	2019-10-25	09:00	FRA	2019-10-25	13:45	CX	288	HKG	2019-10-26	06:50	636	18h 00m	https://w...	0.0
false	BCN	2019-10-25	08:00	LH	1137	FRA	2019-10-25	10:10	FRA	2019-10-25	13:45	CX	288	HKG	2019-10-26	06:50	636	16h 50m	https://w...	0.0
false	BCN	2019-10-25	13:20	KL	1670	AMS	2019-10-25	15:45	AMS	2019-10-25	17:15	KL	887	HKG	2019-10-26	10:20	659	15h 00m	https://w...	30.0
false	BCN	2019-10-25	13:50	LH	1131	FRA	2019-10-25	16:00	FRA	2019-10-25	20:10	CX	282	HKG	2019-10-26	13:25	734	17h 35m	https://w...	30.0
false	BCN	2019-10-25	16:50	LH	1135	FRA	2019-10-25	19:00	FRA	2019-10-25	20:10	CX	282	HKG	2019-10-26	13:25	760	14h 35m	https://w...	30.0
false	BCN	2019-10-25	11:05	KL	1666	AMS	2019-10-25	13:35	AMS	2019-10-25	17:15	KL	887	HKG	2019-10-26	10:20	771	17h 15m	https://w...	30.0

Figura 3.6: ontime Interfaz vuelos de BCN Barcelona a HKG Hong Kong para el día 25-10-2019 y pesos Precio = 100, Duración = 0, Retraso = 10

En el primer caso (Figura 3.5), sólo hemos dado peso al retraso, con lo cual vemos en primer lugar los vuelos con menor retraso, sin ordenación por los demás pesos, por lo que

están “mezclados”. En el segundo caso (Figura 3.6), hemos añadido importancia al precio y al retraso, por lo que en general vemos vuelos sin retrasos en primer lugar pero si un vuelo es más barato según la importancia relativa que le hemos dado, vuelos más baratos pero con más retraso están también al principio de la lista.

A modo de “filtro”, podemos dar peso a una sola variable y ceros a las demás. Si damos pesos diversos, el sistema tampoco actúa como group by múltiple, si no que un orden ponderado será ejecutado.

Requisitos

Para que la aplicación pueda funcionar, se necesita:

- Fichero aeropuertos, B.
- Sesión MongoDB con los datos clima (necesario id key API OpenWeather) ratings actualizados.
- Id key para API flightStats
- Interfaz con Python, acceso a Internet.

Resultado

Las diferentes rutas ordenadas según los pesos introducidos.

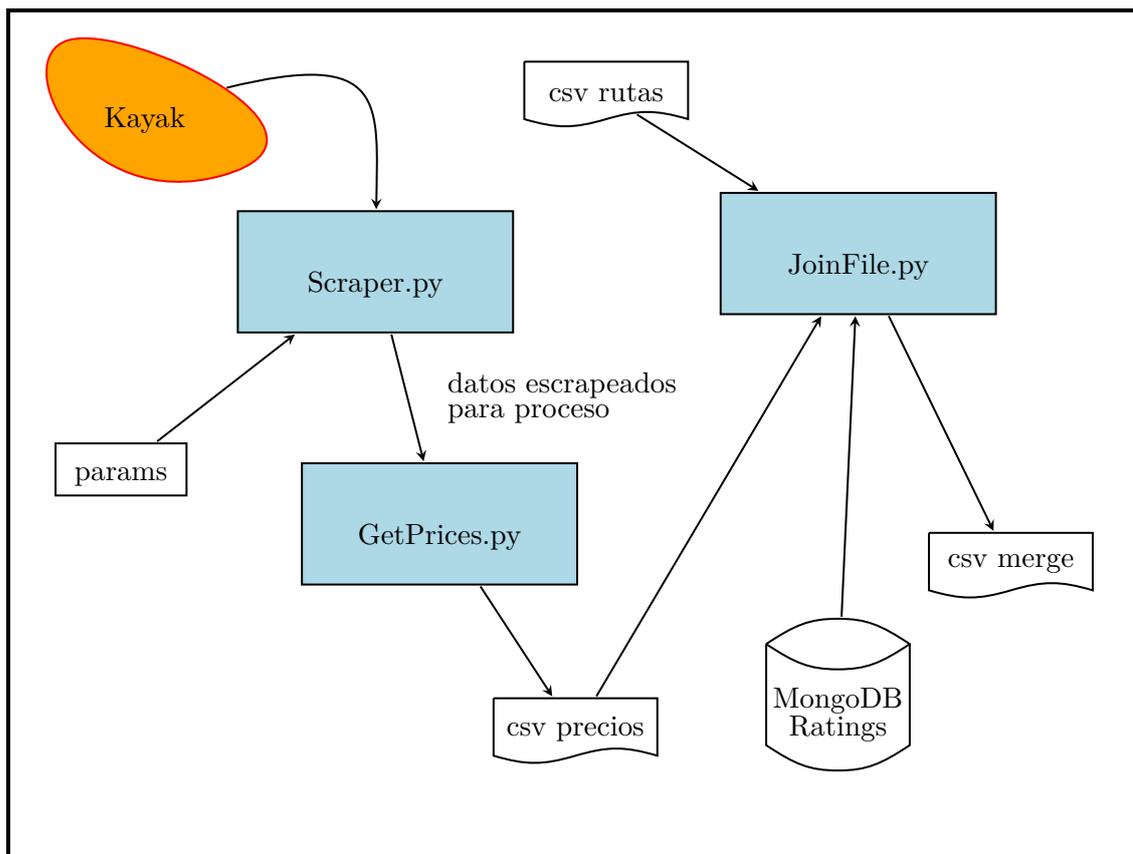
3.3. Módulo búsqueda precios

La principal función de este módulo es acceder a la página web del buscador de vuelos Kayak para poder tener acceso a los precios y a otros elementos que pudieran ser de relevancia en relación a los distintos vuelos proporcionados según tres parámetros principales:

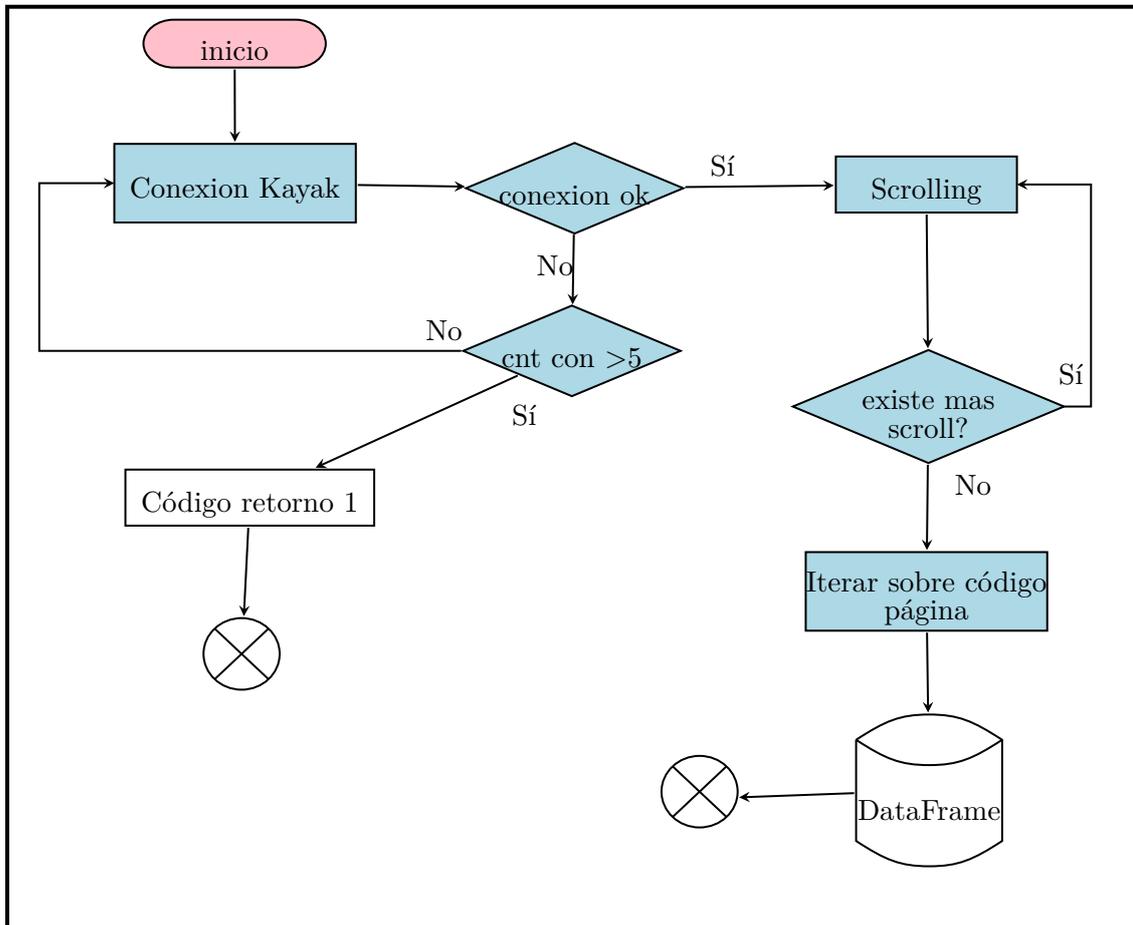
- Aeropuerto de Salida
- Aeropuerto de Llegada
- Fecha de salida

Este módulo se compone de tres ficheros escritos en Python y de dos programas principales que ejecutan el proceso desde el acceso a la web hasta la inclusión de las variables de interés a las rutas y datos climáticos aportados por el módulo de Búsqueda de precios. Finalmente construir un archivos de tipo CSV que aglutine todos los datos para que la ejecución de la predicción de el retraso asociado a los vuelos sea llevada a cabo con éxito.

Diagrama general de este módulo.



3.3.1. Scrapper.py



Este ejecutable Python se encarga del acceso a Kayak a través de una URL creada a partir de los tres parámetros de entrada para acceder a las variables e identificadores principales de cada vuelo y convertir esa información de la web en un Dataset usable por la librería Pandas.

Requisitos:

- Parámetros: aeropuertos salida y llegada en código iata. Fecha salida vuelo yyyy-mm-dd.
- Ejecutable controlador del Driver Selenium para el navegador a usar. Recomendamos Firefox por ofrecer driver navegador sin cabeza.
- Librería BeautifulSoup.

Resultado:

Una vez realizada la conexión a Kayak de forma exitosa y se hallan encontradas las variables necesarias, se procede a la construcción de un objeto tipo DataFrame a partir de éstas y se envían a GetPrices.py para su procesamiento.

La lógica del programa nos permite:

Una vez conectados con la web de Kayak, usando Selenium, el cuál crea un objeto Driver para controlar el navegador de manera oculta (navegador sin cabeza), y una vez que descartamos ventanas de cookies (es necesario hacerlo pues si no, no es posible hacer scroll), se efectúa scrolling sobre la página hasta asegurarnos que no nos dejamos ningún vuelo.

A continuación, se procesa el contenido de la página con BeautifulSoup y cerramos Selenium; en ese momento usamos las funciones de BeautifulSoup para buscar en la estructura de árbol de la página las clases que necesitamos para acceder a los datos de interés para finalmente construir un dataset con ellos. Las variables que recopilamos son:

- Aeropuertos de salida, llegada y la fecha.
- Numero vuelo y aerolínea.
- Precio vuelo.
- Número de vuelos con estatus de codeshare: un codeshare o código compartido ocurre cuando varias aerolíneas gestionan un mismo vuelo, por lo que un vuelo tiene entonces dos números de vuelo oficiales distintos, aunque normalmente se use uno (número de vuelo principal).
- Horarios de salida y llegada.
- Una variable que nos dicte si ese vuelo es directo o posee una escala.
- Aeropuertos intermedios en caso de que haya escala.
- Enlace de compra, se entiende que los enlaces son únicos, es decir para cada vuelo existe un enlace de compra único que nos lleva a la plataforma que gestiona ese vuelo.

Por lo que se refiere a los códigos compartidos, la página los sitúa en un desplegable aparte en forma de lista, por lo que no hay manera de saber (en caso de que por ejemplo el vuelo tenga varias escalas), a qué tramo del vuelo en concreto se refiere cada codeshare por separado; ésto se gestiona en **GetPrices.py**

Testing

Respecto al tiempo de ejecución de Scraper.py este programa depende en gran medida de BeautifulSoup y Selenium mas que de el tiempo de búsqueda de los datos en el árbol de la página web procesada. Al realizar la conexión usando el navegador, aunque sea en oculto, el Driver debe esperar a recibir una respuesta lo cual implica cierto tiempo. De vez en cuando en Selenium es necesario usar tiempos de pausa (time sleep) para dar tiempo al navegador de procesar ya que frecuentemente buscamos elementos en la web que pueden tardar algo en aparecer según lo que tarde el navegador, aunque esto es regulable, contribuye a alargar el tiempo de ejecución.

Otra razón por la que incluir estos tiempos de espera es debido a que si, por el motivo que fuera, la conexión no se realizara con éxito, superado ese tiempo de espera, volveríamos a intentar conectarnos (máximo 5 intentos), si esta conexión se hace sin tiempos de pausa la

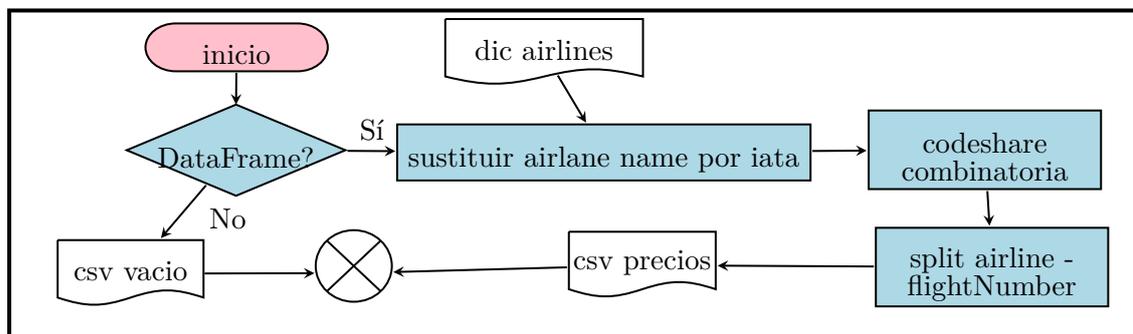
web de Kayak puede pensar que somos un Bot. Esta es la razón principal por la que se ha usado Selenium, se han usado otras librerías clásicas de Python como Requests o URLLib sin embargo no es posible acceder al contenido sin ser bloqueados por la página.

En lo que se refiere a BeautifulSoup, éste se usa cuando hemos podido acceder a la página y ésta haya sido desplegada en su totalidad. Esto tarda un cierto tiempo, ya que se debe hacer clic para seguir haciendo scroll con el consecuente tiempo de espera para que aparezcan nuevos vuelos (lo cual requiere nuevos tiempos de pausa). Por lo que si la ruta escogida es una ruta en la que hay disponibles muchos vuelos, este tiempo puede ser relativamente largo.

BeautifulSoup también debe procesar la información que le proporciona la página por un parser, para luego poder buscar en él. Si la página es grande, el tiempo en procesar esto también lo es. La búsqueda de elementos con BeautifulSoup es rápida, ya que se ejecutan bucles sobre los elementos de interés que en muchas ocasiones no son mas que 300 elementos. Por lo que los datos pasados a GetPrices.py para su procesamiento no suelen tener mas de 300 filas.

En definitiva, dicho programa sirve con éxito a sus propósitos, los tiempos que toma son ajenos a su algoritmo, de hecho aquí no hacemos gestión ninguna, pero sí, las librerías BeautifulSoup y Selenium, necesitan de consumir la latencia del navegador, más las esperas para navegar por los contenidos de la página sin problemas.

3.3.2. GetPrices.py



Este módulo se encarga del procesamiento del Dataset enviado por Kayak a un schema compatible con los datos recibidos del Módulo búsqueda de vuelos para su posterior unión en un nuevo Dataset.

Requisitos:

- Parámetros: aeropuertos salida y llegada. Fecha salida.
- Librería Pandas

Resultado:

Obtención de un CSV precios con el schema adecuado para su unión con los datos recibidos de FlightStats “csv rutas”.

La lógica del programa es la siguiente:

Dado un `dataFrame`, objeto resultado de `Scraper.py` 3.3.1, los principales filtros y modificaciones que aquí se aplican son:

- Substitución de el Nombre de la Aerolínea por su código IATA correspondiente con el uso de un diccionario.
- Aplicación algoritmo de combinatoria (solo aplica a vuelos con una escala y que contengan códigos codeshare).
- Posterior separación de los 4 dígitos de los números de vuelo y su código IATA en columnas separadas. Ejemplo VY1313 para vuelo 1, pasa a ser VY en una columna y 1313 en otra, véase el CSV precios 3.2, que para los dos posibles vuelos en una ruta, genera 4 columnas.

Como filtramos por vuelos directos y con una escala sería razonable pensar que en caso de vuelo con escala pudiéramos tener dos números de vuelo principales y dos codeshares, cada uno de ellos refiriéndose a un tramo del vuelo en particular, pero como no ha sido posible probarlo usamos un algoritmo de **combinatoria** para que, dados un conjunto de número de vuelos principales y números de vuelos codeshare, creamos todos los pares posibles de todos esos números de vuelo para asegurarnos que no se pierde información aunque ello haga que el dataset crezca en número de filas.¹¹

El proceso anterior sólo tendrá lugar en caso de que `Scraper.py` devuelva un `DataFrame`. En caso de no poder establecer conexión con `Kayak`, o no encontrar vuelos en `Kayak`, `Scraper.py` devuelve un código retorno diferente a 0 en lugar de `DataFrame` y `GetPrices` entonces genera un CSV vacío cuyo nombre es específico, pues el nombre de este CSV informa a Java de qué error se trata: `csv.noflight` (hubo conexión pero no hay vuelos), `csv.noconexion` (no pudo establecerse la conexión) o `csv.error` (en otro tipo de “error”).

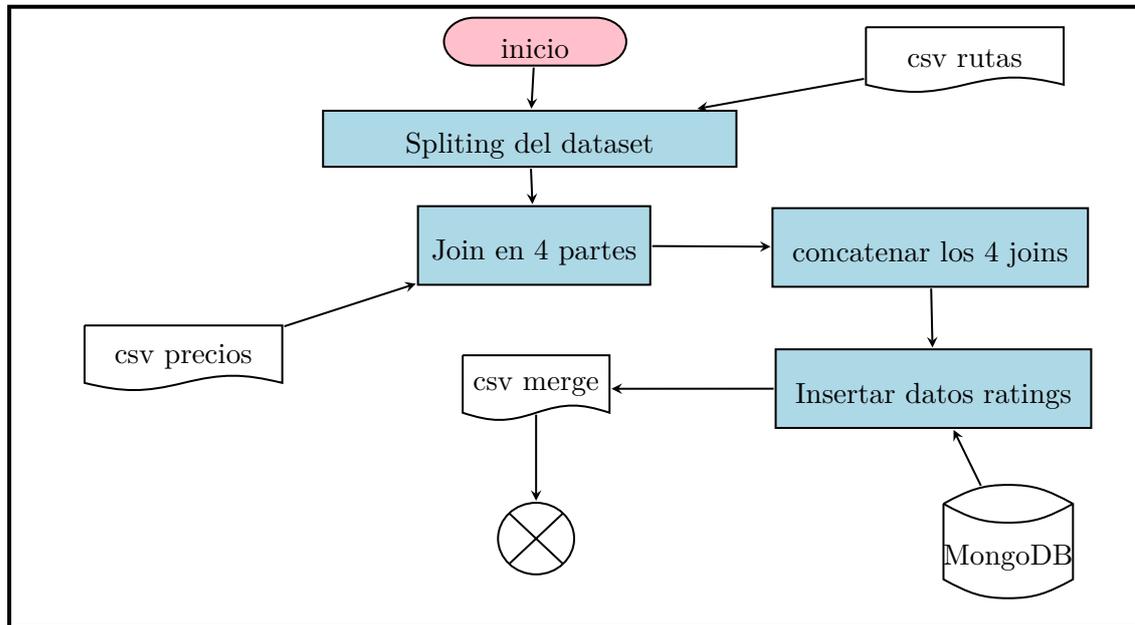
Esta gestión de error, se podría manejar en forma de códigos de retorno e evitar así la latencia de escritura de CSV vacíos y no es nada, al tratarse de creación de ficheros solo. Pero para usar una arquitectura basada en códigos de retorno, se necesita usar sockets o “área de comunicación”, cosa se escapa al propósito de este prototipo.

Testing

`GetPrices.py` se encarga, ahora sí, de gestionar los datos obtenidos por `Scraper.py`. Este hace aumentar el número de filas del dataset en la medida de que aparezcan vuelos que contengan códigos compartidos (codeshares) en vuelos de una escala debido al algoritmo combinatorio antes explicado, por lo que este aumento de filas es más patente en vuelos entre ciudades más o menos lejanas entre sí. En cambio para vuelos de ciudades cercanas los vuelos tenderán a ser directos y este algoritmo no aumentará mucho el número de filas. Aún así, una vez aplicado `GetPrices.py`, el número de filas no suele exceder las 500/600 (ya que existen muchos vuelos que no tienen código compartido siendo vuelos de escala). En cuanto a tiempos de ejecución, son casi despreciables, son bucles en Python sobre 500/600 filas sin una gestión que implique muchos cálculos.

¹¹Si se supiera a qué vuelo de la ruta corresponde un codeshare, esto no sería necesario.

3.3.3. JoinFile.py



La utilidad de este módulo de Python es unir los datos recibidos de Flightstats (CSV rutas) con la información obtenida desde kayak (CSV precios) ambos en forma de CSV. Así como añadir datos de rating y preparar un CSV resultado con toda esta información unificada en lo que hemos llamada “csv merge”.

Requisitos:

- Parámetros: aeropuertos salida, llegada y fecha salida.
- CSV de rutas, CSV de precios.
- Sesion MongoDB.
- Librería Pandas.

Resultado: CSV merge, union de ambos CSV. Devuelto a Java sobre el cual se aplica la predicción.

Más en detalle, la lógica del programa es:

Ejecutar la unión (en este caso, “Join”) de el csv precios con los datos recibidos de el módulo Búsqueda de Vuelos con Flightstats también en forma de .csv rutas.

Para poder efectuar esa unión, dividimos los datos de Flightstats (csv rutas) en 4 partes, a saber:

- Vuelos de una escala sin nulls.
- Vuelos de una escala con nulls.

- Vuelos directos sin nulls.
- Vuelos directos con nulls.

La razón por la cual es necesario hacer esta distinción es principalmente por el hecho de tener codeshares en el dataset rutas sin número de vuelo y por otra parte, la información que se escrapea de Kayak, en ocasiones puede ser incompleta y disponer de datos importantes para hacer join sin ser informados. Es, por tanto, necesario, considerar diferentes campos para hacer join evitando el hecho de perder coincidencias entre los datasets por el hecho de tener un campo a null cuanto el resto que puede caracterizar un vuelo están bien alimentados.

Para hacer los joins, usamos las siguientes variables:

- Códigos IATA y los dígitos numéricos que configuran el número de vuelo junto con los horarios de salida y llegada del vuelo (Vuelos de una escala sin nulls y vuelos directos sin nulls). Notemos que para una fecha dada, el vuelo queda caracterizado por código compañía + número vuelo y si esta información la tenemos, bastaría joins por dichos campos.
- Códigos IATA junto con horarios de salida y llegada (vuelos de una escala con nulls y vuelos directos con nulls), ya que en esta situación desconocemos los números de vuelo en Flightstats (el problema de codeshares sin número vuelo informado, y el hecho de que un vuelo en Kayak puede aparecer con cualquier código de los que existen en la lista codeshares)¹²

En el segundo tipo de join, el número de vuelos candidatos en el dataset resultante aumenta sustancialmente ya que al hacer el Join por menos campos aumentamos potencialmente el número de registros (algunos de ellos duplicados), sin embargo es necesario hacerlo para intentar captar aquellos vuelos que son el mismo pero en flightStats nos viene sin número vuelo informado.

En este punto tenemos los joins completados, entonces procedemos a borrar algún posible duplicado (por el hecho de hacer joins sin campos que caracterizan 100% los vuelos en segundo join, y por usar combinatoria al no saber a qué tramo del vuelo corresponde el codeshare), un duplicado a este nivel es un vuelo con un **HREF** ya existente en el dataset.

Y finalmente, antes de escribir el resultado del join en CSV, añadimos las variables de Ratings que tenemos almacenadas en MongoDB.

Testing

Para JoinFile.py, aunque en su núcleo se encuentren los Joins y estos sean caros de hacer, el tiempo que tarda es bastante pequeño ya que el número de filas de los datasets a unir es relativamente pequeño (el CSV de rutas suele ser mas numeroso que el obtenido de Kayak, CSV precios, una cantidad común en los tests han sido 1200 para el primero y 400 para el segundo aproximadamente) y por tanto, por lo general, no se deberá usar con datasets mucho

¹²En la versión de pago, estos problemas no existirían, o al menos la compañía flightStats asegura informar todos los campos.

mayores ya que para una ruta y una fecha, los vuelos no pueden ser muchos.

El resultado de estos Joins es un dataset de menor envergadura que el obtenido de Kayak y en numerosas ocasiones sale de bastante menor tamaño, posibles razones para esto es que los datos obtenidos del módulo Búsqueda de Vuelos son datos de período de prueba (ya que este servicio es de pago) y los datos obtenidos de Kayak tan solo sea una selección de la cantidad total de vuelos para una cierta ruta.

En definitiva, dicho programa sirve para sus propósitos.

3.3.4. Herramientas principales

Pandas

Se ha usado principalmente en la fase correspondiente al preproceso necesario de los datos recibidos de Kayak y hacerlos apropiados para su acople con los datos recibidos de Flightstats. Así como de apoyo al proceso de unión de estos con los anteriores. En general nos permite la manipulación básica de cualquier dataset.

BeautifulSoup

Dado que la información descargada de Kayak nos llega en formato página web. Esta librería es la principal encargada de condensar esa información en un marco de trabajo tipo "Dataframe" que Pandas pueda manipular. Se pueden buscar todo tipo de elementos en la estructura árbol de la página.

Selenium

Esta librería se encarga de interactuar con la web controlando directamente el navegador, en este caso Firefox, a través del uso de un Driver, el cual se genera a través de un ejecutable guardado en nuestro proyecto, el cual a su vez dependerá del navegador que queramos hacer servir. Su principal propósito para el que fue diseñado fue la implementación de tests para aplicaciones web para interactuar con estas como si se tratara de un usuario.

4. Data Analysis

4.1. Introducción

En la literatura podemos encontrar ciertos artículos recientes que tratan la cuestión del retraso de vuelos. Los podemos clasificar en dos: los que buscan conocer las causas de los retrasos dado que suponen un gasto relevante para aeropuertos y aerolíneas, y los que buscan, como nuestro caso, predecir futuros retrasos. Estos últimos, por lo general, usan menos cantidad de variables ya que algunas no se pueden prever con antelación a los vuelos.

Diversas técnicas se han usado en ambos enfoques, desde técnicas más clásicas de estadística a técnicas de Machine Learning. La siguiente gráfica tomada del artículo Alice y cols. (2017)¹ muestra que el abanico de técnicas usadas por diversos autores es amplio:

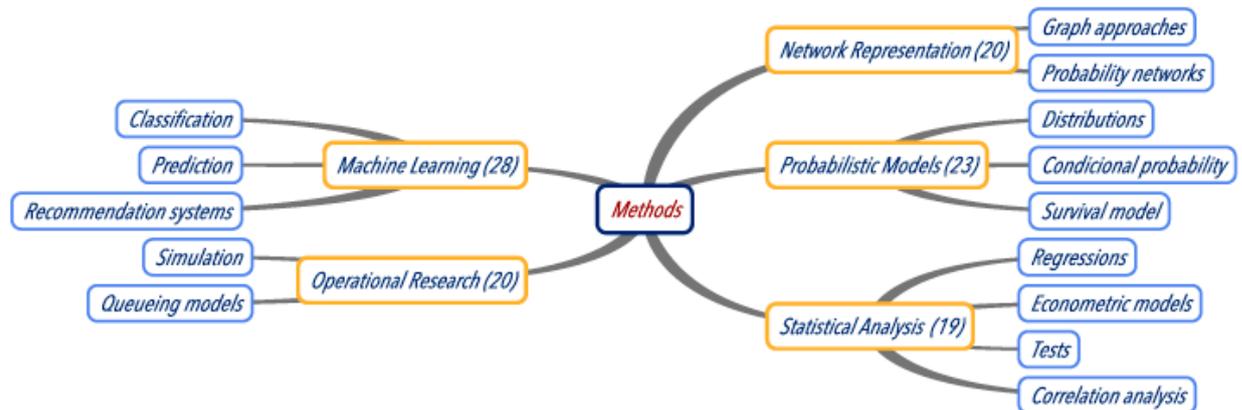


Figura 4.1: Diferentes técnicas usadas en la literatura para tratar el problema de retraso en vuelo

Trabajos realizados por Anish y Aera (2017) se enfocan en el estudio de las causas de los retrasos. Partiendo de datos ofrecidos por el “Bureau of Transportation Statistics”, donde se encuentran especificaciones del tipo de retraso que sufren los vuelos², elaboran modelos de regresión lineal múltiple, árboles de decisión y Random Forest, concluyendo mejores resultados con éstos últimos, siendo las causas más significativas la propia aeronave, el mantenimiento, el retraso acumulado de escalas anteriores, etc.

En la misma línea, el trabajo realizado por Martina y cols. (2017) busca conocer qué variables están más correlacionadas con los retrasos y los tipos de éstos (véase la tabla 4.1). Para ello usan tests χ^2 sobre tablas de contingencia, concluyendo que existe dependencia con la variable mes y momento del día, así como la atribución de que el gran porcentaje de retrasos

¹Se recomienda su lectura para tener un espectro de los trabajos ya realizados en la cuestión de los retrasos en vuelos.

²Únicamente datos de vuelos entre aeropuertos de Estados Unidos con compañías americanas.

se debe al tipo “R” es decir, al retraso debido a la aeronave - retrasos en sus vuelos precedentes.

Stefan y Christian (2019), incluyen variables temporales relativas a vuelos, como fecha de salida, llegada, aeropuerto, compañía y variables relativas al clima, centrándose en estas últimas para analizar la influencia del clima en los retrasos que sufren los vuelos. Concluyen que únicamente condiciones climatológicas algo extremas pueden tener un efecto significativo.

Charles y Michael (2013), enfocan el problema desde un punto de vista de congestión de los aeropuertos y del tráfico aéreo. Usando técnicas de programación estocástica y programación entera multiobjetivo, buscan modelos que permitan ofrecer, teniendo en cuenta predicciones del clima en el aeropuerto de llegada, soluciones en forma de horarios de salida óptimos, para evitar congestiones en el aeropuerto de llegada. Es decir, buscan minimizar los retrasos en salida debido a congestiones en la llegada.

Finalmente, un trabajo con técnicas de Machine Learning, con el objetivo de predecir los retrasos de vuelos futuros, enfoque similar al que aquí desarrollaremos, sería el trabajo de ?. Usando Random Forest generan modelos para predicción (clasificación binaria) a 2-24h vista, con error promedio en test de 0.19. Consideran variables temporales como mes, día, momento del día en que se desarrolla el vuelo, y variables espaciales, como los aeropuertos involucrados. Un estudio por aeropuertos y por rutas es considerado. Para la variable momento del día, se hace uso de K-means para generar 6 clusters en que categorizan dicha variable. También un modelo de regresión es introducido para estimar el posible retraso del vuelo.

Codes	Explanation
AIC	Operational reasons of airline
PB	Delay because of passengers and their baggage
ARH	Delay caused during the aircraft handling by suppliers - handling, fuel, catering
TAE	Delay caused by technical maintenance or aircraft defect
FOC	Delay caused by operational control and crew duty norms
ATFMR	Delay caused by air traffic control
AGA	Delay caused by airport limitation
R	Reaction codes - Delay caused by delay of previous flight
MISC	Specific delay, can't be included to categories

Tabla 4.1: Clasificación de las causas de retrasos aereos. Fuente: Martina y cols. (2017)

En el presente capítulo, nuestro objetivo es hacer un análisis de diferentes modelos de clasificación, así como explorar los datos de los que disponemos con el objetivo de ofrecer un algoritmo robusto, sin overfitting y con accuracy aceptable para poder confiar las predicciones futuras. Dichas predicciones forman una parte fundamental de la idea de negocio, y servirán para ofrecer vuelos a los usuarios, con predicción del posible retraso en su vuelo.

4.2. Data set, desde origen del dato hasta su explotación

Recordemos que los datos que usamos provienen de diferentes fuentes: información temporal y espacial referente a los vuelos de la API routeStatus, estadísticas sobre los vuelos de la API routeRatings y datos del clima de la API OpenWeather. Véase la sección 3.1 donde se explican los programas encargados de acceder a los datos y almacenarlos, ahí dispondrá de referencias a las respuestas de las APIs y podrá conocer la estructura de los documentos. Se muestran a continuación los datos que se preparan usando el programa Dataset.jar 3.1.5 y que ofrecemos en formato CSV para el modelado y el análisis.

1. **departureAirportFsCode_v1**. *String*, código iata del aeropuerto salida. Ejemplo BCN.
 2. **arrivalAirportFsCode_v1**. *String*, código iata del aeropuerto salida. Ejemplo MAD.
 3. **year**. *Integer*, año salida vuelo. Ejemplo 2019.
 4. **month**. *Integer*, mes salida vuelo. Ejemplo 11.
 5. **day**. *Integer*, día salida vuelo. Ejemplo 8.
 6. **carrierFsCode_v1**. *String*, código iata aerolínea. Ejemplo IB.
 7. **flightNumber**. *String*, número vuelo. Ejemplo 1631.
 8. **fsCodeFlightNumber**. *String*, concatenación de las dos variables anteriores. Ejemplo IB1631.
 9. **departureDate**. *String*, fecha y hora salida local en formato “ISO-8601 format. yyyy-MM-dd'T'HH:mm:ss.SSS”. Ejemplo 2019-11-08T16:30:00.000.
 10. **departureDateUtc**. *String*, fecha y hora salida UTC en formato “ISO-8601 format. yyyy-MM-dd'T'HH:mm:ss.SSSZ”. Ejemplo 2019-11-08T14:30:00.000Z.
 11. **arrivalDate**. *String*, fecha y hora llegada local en formato “ISO-8601 format. yyyy-MM-dd'T'HH:mm:ss.SSS”. Ejemplo 2019-11-08T17:55:00.000.
 12. **arrivalDateUtc**. *String*, fecha y hora llegada UTC en formato “ISO-8601 format. yyyy-MM-dd'T'HH:mm:ss.SSSZ”. Ejemplo 2019-11-08T15:55:00.000Z.
 13. **flightType**. *String*, tipo de vuelo, para pasajeros, comercial, de correo etc. Ejemplo J “vuelo pasajeros servicio normal”.
 14. **departureGateDelayMinutes**. *Integer*, retraso en salida dado en minutos. Ejemplo 17.
 15. **temp_v1**. *Double*, temperatura entorno al aeropuerto salida en kelvin. Ejemplo 304.2
 16. **pressure**. *Double*, presión atmosférica entorno al aeropuerto salida en *hPa*. Ejemplo 1014.39
 17. **weather_v1**. *String*, condición climática entorno al aeropuerto salida. Ejemplo Rain.
-

18. **wind_speed**. *Double*, velocidad viento entorno al aeropuerto salida en *m/s*. Ejemplo 2.42
19. **windDeg_v1**. *Double*, dirección viento entorno al aeropuerto salida en grados meteorológicos. Ejemplo 166.682
20. **humidity**. *Double*, humedad entorno al aeropuerto salida en *%*. Ejemplo 51.
21. **ontime**. *Integer*, número de observaciones³ que el vuelo sale a su hora programada o con retraso inferior a 15 minutos. Ejemplo 12.
22. **late15**. *Integer*, número de observaciones que el vuelo se retraso más de 15 minutos pero menos de 29. ejemplo 8.
23. **late30**. *Integer*, número de observaciones que el vuelo se retraso más de 30 minutos pero menos de 44. Ejemplo: 4.
24. **late45**. *Integer*, número de observaciones que el vuelo se retraso más de 45 minutos. Ejemplo 2.
25. **delayMean**. *Double*, media en minutos de retrasos acumulados desde que FlightStat dispone de datos sobre dicho vuelo. Ejemplo 16.0
26. **delayStandardDeviation**. *Double*, desviación típica de retrasos acumulados. Ejemplo 4.0
27. **delayMin**. *Integer*, retraso mínimo observado en minutos. Ejemplo 5.
28. **delayMax**. *Integer*, retraso máximo observado en minutos. Ejemplo 72.
29. **allOntimeCumulative_v1**. *Double*, porcentaje que expresa el ranking que ocupa el vuelo en comparación con los demás en cuanto a salir en su hora. Ejemplo 0.8
30. **allOntimeStars_v1**. *Double*, en una escala de 0 a 5, rating de salida a su hora comparado con el resto de vuelos.
31. **allDelayCumulative_v1**. *Double*, porcentaje que expresa el ranking que ocupa el vuelo en comparación con los demás en cuanto a salir con retraso. Ejemplo 0.71
32. **allDelayStars_v1**. *Double*, en una escala de 0 a 5, rating de salida con retrasos comparado con el resto de vuelos.

Las variables [1-14] se obtienen de la colección correspondiente a la API `routeStatus`; las variables [15-20] de la colección relativa a la API `airportWeather`, mientras que las variables [21-32] se obtienen a su vez de la colección relativa a la API `routeRatings`.

Cabe destacar que dichas API's ofrecen más datos, pero algunos resultan poco relevantes para el análisis y por tanto los descartamos. Hay variables que no usaremos en los modelos de entrenamiento, como por ejemplo `flighttype`, ya que en más del 90% de los casos el tipo de vuelo corresponde a vuelos "pasajeros de servicio normal" "J", que no aportan nada al modelo, pero que nos pueden servir como información para otros propósitos en relación

³Las observaciones son observaciones acumuladas desde que FlightStats dispone de información sobre el vuelo.

al CSV. Variables como “Taxi-out” y “Taxi-in”, que nosotros descartamos, otros trabajos incluidos en las referencias las usan con el objetivo de analizar los retrasos en “block time”, es decir, los retrasos en la duración que la compañía estima para el viaje, cosa que aquí no valoramos. Nos centramos en retrasos en salida. También descartamos variables como el tipo de aeronave o la terminal del aeropuerto; consideramos que por lo general, hoy en día las aeronaves habituales (de transporte de pasajeros) suelen ser similares, y no todos los aeropuertos disponen de diferentes terminales, con lo cual, no aportarían información relevante de cara los modelos. Los modelos que pretendemos usar son modelos generales, que pretenden englobar los aeropuertos y aerolíneas de todo el mundo. Los trabajos que se pueden encontrar en la literatura se centran en regiones o conjunto de aerolíneas; en dichos casos, conocer tipo aeronave, terminales, etc., puede aportar más información para detectar problemas de mantenimiento en aerolíneas y gestión de aeropuertos.

4.3. Preprocesamiento

En cuanto al preprocesamiento de datos, parte fundamental de todo análisis, se exponen a continuación las transformaciones llevadas a cabo. Se realiza en el programa Model.jar 3.1.6 y para ello disponemos de las siguientes funciones:

- **Función categorizeDate.** Nos permite categorizar las fechas de salida y llegada del vuelo (departureDate y arrivalDate). El mes del año toma 12 categorías “Jan”, “Feb”, ..., “Dec”. El día toma 7 categorías “Mon”, “Tue”, ..., “Sun”. También se discretiza la hora del día en 24/3 categorías.
 - **Función categorizeDelayAndWeather.**
 - Categorizar la variable departureGateDelayMinutes en 2 clases, No retraso si es menor a 15 minutos y sí hay retraso en otro caso. En caso de multiclases, las clases son: [0,15],[15,30],[30,45],[45,60],[60,Integer.MaxValue].
La variable resultante de dicha transformación es la variables respuesta “Label”.
 - Categorizar la variable delayMean_v1 en las mismas categorías que la anterior. Así tenemos una frecuencia de vuelos retrasados implícitamente.
 - Categorizar la variable temp_v1 en tres categorías que vienen a representar “mucho frío”, “normal”, “Mucho calor”, buscando los extremos en temperatura que pueden afectar de algún modo al normal funcionamiento de las aeronaves e instalaciones.
 - Categorizar la variable windDeg_v1 en 18 categorías, una partición del globo terráqueo.
 - Categorizar allOntimeStars_v1 en 10 categorías, desde 0 hasta escala 5 de saltos de 0.5 puntos. Son las diferentes puntuaciones en las que la API routeRatings clasifica el rating de un vuelo.
 - Categorizar allDelayStars_v1 de la misma manera que la variable allOntimeStars_v1
 - **Función Indexer.** Un proceso técnico más propio del análisis que nos permite indexar las variables categóricas en formato numérico. Así, el objeto OneHotEncoder puede
-

servirnos a la hora de seleccionar las variables a considerar en el modelo .

Cabe destacar aquí que es importante usar `StringIndexerModel` en lugar de `StringIndexer`. Supongamos que tenemos la variable `departureAirportFsCode = "BCN"`, al indexar `BCN=1.0`, `MAD=2.0` y así con todos los aeropuertos. Una vez guardado el modelo, a la hora de usarlo nuevamente para inferir vuelos futuros, debemos volver a hacer todo el preproceso de datos (para así poder aplicar el modelo al vuelo en cuestión), así como el proceso de indexación. En caso de usar `StringIndexer`, no podemos guardar el indexado que se hizo durante el entrenamiento y por tanto perdemos esa relación entre categoría e índice. Puede pasar que si el vuelo a predecir sale de MAD, su índice con `StringIndexer` sería 1.0 cuando ese índice en el momento de entrenar correspondía a BCN. Esto es un problema. Otro es que normalmente predecimos sobre pocos vuelos, con lo cual no tendremos todos los aeropuertos representados y el vector de índices será de diferente dimensión que el usado en el entrenamiento, lo que genera un error en los cálculos y el programa falla. `StringIndexerModel`, permite guardar ese esquema de índices, y en el momento de predecir, lo que hacemos es cargar esos índices y usarlos, evitando así los dos problemas anteriores. Nuestras variables tendrán el índice que les corresponde y el tamaño del vector sería el correcto.⁴

- **Función `featureSelection`.** Nos permite seleccionar las features, en caso de modelo sin datos del clima, no se seleccionan las variables provenientes de `airportWeather`.

1. `departureAirportOneHot`
2. `arrivalAirportOneHot`
3. `carrierFsCodeOneHot`
4. `tempOneHot`
5. `humidity`
6. `pressure`
7. `wind_degOneHot`
8. `weatherOneHot`
9. `wind_speed`
10. `departureMonthOneHot`
11. `arrivalMonthOneHot`
12. `departureDayOneHot`
13. `arrivalDayOneHot`
14. `departureDayMomentOneHot`
15. `arrivalDayMomentOneHot`
16. `delayMeanLabelOneHot`
17. `allOntimeCumulative_v1`

⁴Este problema que parece simple una vez conocido el cambio, nos ha llevado mucho tiempo de corregir y cambiar `StringIndexer` por `StringIndexerModel`, gracias a nuestro tutor Alberto Gutiérrez por sugerirnos el cambio a `StringIndexerModel` Object.

18. allOntimeStarsOneHot
19. allDelayCumulative_v1
20. allDelayStarsOneHot

- **Función balanceDataset.** En caso de algoritmos distintos a RandomForest, podemos indicar al modelo cómo de desbalanceados están los datos con la columna *classWeightCol*. Ésta función nos permite calcular el desbalanceo de las clases. En caso de RandomForest, que no lleva implementado ningún algoritmo capaz de tener en consideración el posible desbalanceo del dataset, se efectúa un balanceo “manual”, es decir, dividimos el dataset en las diferentes clases y suprimimos registros de manera random de aquella clase que más representada está, hasta equilibrar todas las clases.

En este punto ya tenemos un Dataset<Row> y podemos aplicar modelos, bien de regresión (en cuyo caso no se categoriza la variable de respuesta) bien de clasificación.

Observación. Gran parte de las transformaciones son “naturales”, como por ejemplo la variable fecha y hora. Considerarlos numéricas no aportarían información relevante, ni son en realidad continuas; por tanto, categorizarlas de alguna manera es necesario. Otras transformaciones, como discretizar la temperatura, la orientación del viento, etc, fueron hechas después de entrenar modelos y buscar mejorar la accuracy de los mismos.

En cuanto a la limpieza de los datos, éstos por lo general disponen de la información básica que aquí usamos, por lo que aquellos vuelos con algún dato sin informar, son suprimidos⁵.

4.4. Técnicas y algoritmos

Como se puede leer en la literatura, en muchas ocasiones, modelos de regresión y clasificación son considerados para tratar el tema de “flight delays”, donde Random Forest obtiene mejores resultados. Por lo general Random Forest ha mostrado buenos resultados en clasificación y el cómputo es relativamente rápido. Además, ofrece la posibilidad de computar la relevancia de las variables. Aquí probaremos el comportamiento de dicho método, pero también otros modelos, pues como mencionamos anteriormente, el programa Model.jar 3.1.6 nos permite entrenar todos los modelos que ofrece la librería Spark MLlib en clasificación. Genera un CSV con los resultados de los diferentes modelos, la accuracy, el recall de cada clase, así como el tiempo total invertido en el entrenamiento, lo que nos servirá para hacer comparativas y concluir posibles mejoras para un modelo final que nos sirva para el negocio.

En la siguiente tabla podemos ver los resultados sobre el CSV generado en la sección anterior.

⁵Podrían suprimirse únicamente aquellos registros que no disponen de información en alguna variable involucrada en los cálculos, pero es una simplificación técnica borrar los registros con cualquier variable con valor nulo. No se perderían muchos registros.

date	model	weather	accuracy	recall_0	recall_1	timeSeg	Mean
29/09/2019	GBTC	N	67.353	70.846	63.705	1708	67.301
29/09/2019	GBTC	Y	67.117	70.847	63.290	2046	67.085
23/09/2019	LR	Y	66.517	66.142	68.111	431	66.923
23/09/2019	OneVsRestLR	N	66.295	65.973	67.677	59	66.648
29/09/2019	OneVsRestLR	Y	66.381	66.265	66.892	58	66.513
23/09/2019	LinearSVM	N	68.925	71.511	57.988	316	66.141
29/09/2019	LinearSVM	Y	65.209	64.557	68.057	1769	65.941
23/09/2019	OneVsRestSVM	Y	65.240	64.681	67.625	337	65.849
23/09/2019	LinearSVM	Y	64.490	63.384	69.226	615	65.700
23/09/2019	NB	N	65.205	64.798	66.958	7	65.654
29/09/2019	OneVsRestSVM	N	64.964	64.375	67.528	341	65.622
28/09/2019	NB	Y	65.009	64.615	66.721	12	65.448
23/09/2019	RF_CV	Y	64.477	66.640	62.289	11189	64.469
28/09/2019	RF	Y	64.086	69.521	58.495	153	64.034
29/09/2019	RF	N	63.906	65.924	61.844	138	63.891
23/09/2019	RF	N	63.908	69.911	57.797	113	63.872
04/10/2019	LR	N	81.770	100.0	0.0	200	27.257

Tabla 4.2: Accuracy de los diferentes modelos en clasificación binaria.

Obtenemos como mejor modelo Gradient-boosted tree classifier (GBTC). Es una técnica de ensamble learning usando árboles de decisión, pero se diferencia de Random Forest en el hecho de que los árboles usados son dependientes. Cada iteración añade un árbol a los resultados anteriores, en lugar de entrenar árboles de manera independiente y agrupar el resultado al final como haría Random Forest⁶.

Podemos ver la importancia de las variables que ofrece GBTC, usando la librería feature-importance-helper⁷ y que el programa guarda en un fichero. Podemos ver que el top-10 de variables más importantes es:

```
FeatureInfo [rank=0, score=0.06962179171555277, name=allOntimeCumulative]
FeatureInfo [rank=1, score=0.0281172860774523, name=allDelayCumulative]
FeatureInfo [rank=2, score=0.024168932285160354, name=departureMonthOneHot_Sep]
FeatureInfo [rank=3, score=0.021725064708990993, name=departureDayOneHot_Fri]
FeatureInfo [rank=4, score=0.019842163732681984, name=carrierFsCodeOneHot_6E]
FeatureInfo [rank=5, score=0.017314051698011642, name=departureDayMomentOneHot_d2]
FeatureInfo [rank=6, score=0.01660911890261435, name=departureDayOneHot_Sat]
FeatureInfo [rank=7, score=0.016512391297698686, name=carrierFsCodeOneHot_JT*]
FeatureInfo [rank=8, score=0.016405364050510757, name=carrierFsCodeOneHot_U6]
FeatureInfo [rank=9, score=0.013559233016551955, name=departureAirportOneHot_CAN]
```

Es decir, aparecen levels de variables como allOntimeCumulative, allDelayCumulative, departureMonth, departureDay, carrierFsCode, departureDayMoment y departureAirport.

⁶En el siguiente link, encontrará un curso de la Universidad de Washington dedicado a este modelo y que recomendamos <https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf>.

⁷<https://github.com/riversun/spark-ml-feature-importance-helper>

El hecho de que las variables estadísticas `allOnTimeCumulative` y `allDelayCumulative` aparezcan como las más importantes, nos sugiere la siguiente cuestión. ¿El modelo aporta mejoras respecto a una clasificación por función discriminante usando las estadísticas de `flightRatings API`? Es decir, nuestro modelo es mejor comparado con un modelo que usa “`DelayMean`” de `Ratings` y una función discriminante f tipo si $f(x) > 15$ predecir retraso, y no en otro caso. Pues este modelo simple tiene el siguiente output:

```
Algorithm using delayMean and discriminate function
3554.0 10048.0
1719.0 11587.0
Accuracy %= 56.26951092611863
Recall label 0.0 = 26.12851051315983
Recall label 1.0 = 87.08101608297008
```

Observamos que efectivamente supone una mejora; es más, podemos decir que los datos usados en nuestros modelos son algo sesgados en el sentido de que sólo involucran algunos días, por limitación de acceso a datos. Con más datos, el resultado sería aun mejor.

Curiosamente (o quizás no), ya que otros trabajos de la bibliografía venían anunciándolo, las variables del clima no aparecen en la cabecera de variables importantes, debido a que, por lo general, sólo afectan las condiciones climatológicas con elevada significancia, pues la primera variable referida al clima que aparece es `weathermean_snow`.

No se ha realizado ningún análisis de reducción de dimensionalidad como PCA, dado que 18 variables que usamos en el modelo final no tienen fuerte correlación entre ellas y son comprensibles. Usar PCA conlleva crear nuevas variables a interpretar para bajar la dimensionalidad cuanto ésta es elevada. En este caso no consideramos que 18 variables sean muchas. Veamos un análisis de correlación entre variables, en este caso numéricas, para ver si están muy correladas con otras y aportan algo al modelo. También un test χ^2 de Pearson nos puede servir para conocer las variables más correladas con la variable respuesta.

	temp	pressure	wind_speed	windDeg	humidity	delayMean	allOnTimeC	allOnTimeS	allDelayC	allDelayS
temp	1	-0,314	0,046	-0,075	-0,443	0,046	-0,053	-0,053	-0,09	-0,09
pressure	-0,314	1	-0,203	-0,077	-0,006	0,023	0,026	0,026	-0,019	-0,02
wind_speed	0,046	-0,203	1	0,007	0,034	0,009	-0,077	-0,077	-0,014	-0,014
windDeg	-0,075	-0,077	0,007	1	-0,008	-0,001	0,003	0,003	0,011	0,011
humidity	-0,443	-0,006	0,034	-0,008	1	-0,044	0,067	0,067	0,096	0,096
delayMean	0,046	0,023	0,009	-0,001	-0,044	1	-0,285	-0,285	-0,739	-0,739
allOnTimeC	-0,053	0,026	-0,077	0,003	0,067	-0,285	1	1	0,389	0,389
allOnTimeS	-0,053	0,026	-0,077	0,003	0,067	-0,285	1	1	0,389	0,389
allDelayC	-0,09	-0,019	-0,014	0,011	0,096	-0,739	0,389	0,389	1	1
allDelayS	-0,09	-0,02	-0,014	0,011	0,096	-0,739	0,389	0,389	1	1

Tabla 4.3: Tabla de correlaciones entre las variables numéricas

Encontramos relaciones fuertes entre pares de variables como `allOnTimeStarts` & `allOnTimeCumulative`, `allDelayStarts` & `allDelayCumulative` con correlación 1, por lo que aportan la misma información. En consecuencia, se modifica el modelo para descartar las variables `allOnTimeStarts` y `allDelayStarts`. Otras relaciones fuertes se hallan entre `DelayMean` & `allDelayCumulative`, una relación negativa, pues a más retrasos, menos puntuación. Se podría

descartar la variable `delayMean` del entrenamiento del modelo, sin embargo es una información relevante que mantenemos, sobre todo de cara a futuras evoluciones, donde podamos necesitar de dicha variable. Creemos que el mantenimiento del código es también algo a tener en cuenta.

	<code>depAirpor</code>	<code>arrAirpor</code>	<code>carFsCode</code>	<code>depMonth</code>	<code>arrMonth</code>	<code>depDay</code>	<code>arrDay</code>	<code>depDayMoment</code>	<code>arrDayMoment</code>
pValues	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
degreesOfFreedom	1372	1391	560	2	2	6	6	7	7
statistics	13423,004	11158,960	17112,130	1799,099	1790,308	1758,606	1348,148	5412,144	4533,337

Tabla 4.4: χ^2 test independencia variables categóricas vs variable respuesta

Vemos que en todos los casos el p-valor es casi siempre nulo, con lo cual podemos aceptar la hipótesis alternativa de que existe dependencia entre las variables categóricas y la variable respuesta.

Nota: para un análisis descriptivo de los datos, véase anexo C.

4.5. Resultados y conclusiones

Se han usado diversos modelos de clasificación, obteniendo mejores resultados con Gradient-boosted tree classifier. Modelos con y sin datos del clima. Los modelos con datos del clima tienen limitación a cinco días vista en sus predicciones, dado que no existen datos más allá de 5 días (en los datos gratuitos). Cabe destacar que los datos del clima no aportan mucho al modelo, dado que por lo general, solo afectan al transcurso normal en la salida de un vuelo aquellas condiciones climatológicas complicadas; cierto es también que no se han tenido en cuenta las condiciones meteorológicas del aeropuerto de llegada en torno a la fecha prevista de llegada (en ocasiones malas condiciones climatológicas en el aeropuerto de llegada, hace que el vuelo salga más tarde). Sería una “mejora” a tener en cuenta, aunque nuevamente solo afectarían las condiciones extremas como por ejemplo nieve “`weather_main = snow`”, que obtuvimos como variable con score no nulo en “Random Forest feature importance”. En la misma línea, uno esperaría que los datos del clima tuvieran más importancia; hay que recordar que en sí mismos ya se tratan de una predicción, con lo cual se acumularía el error. No obstante, últimamente los datos climatológicos son bastante acertados.

Aparte de la clasificación binaria de la que hemos hablado antes, hemos usado en un primer intento clasificación multiclase: (0, 14] min. sería clase 0.0, [15, 29] min. clase 1.0, [30, 44] min. clase 2.0, [45, 59] clase 3.0 y el resto, retraso de más de una hora sería clase 4.0. Pero los resultados no han sido prometedores. Otro modelo que se planteó en esta línea es la clasificación binaria, y dentro de los vuelos clasificados con retraso, volver a aplicarles un modelo de clasificación o regresión.⁸

En este tipo de problemas abundan variables categóricas con muchos levels, como aeropuertos, aerolíneas, mes del año, días de la semana, etc. entre otras, y por tanto, para que un

⁸No ha sido posible evaluar esta vía por falta de tiempo.

modelo no quede sesgado a una parte de los datos, necesitan una gran cantidad para poder disponer de muestras de todas las categorías y sus levels. Esta es una razón por la cual es mejor usar Spark MLlib, pues aunque los datos que hemos usado (más de 350.000 vuelos) se pueden tratar con las librerías de R o Python sin uso de librerías de Spark, en realidad los modelos necesitarían muchos más datos, y así un entrenamiento con paralelismo sería más conveniente.

Por otra parte, dado que es necesario disponer de muchos datos, hay que tener en cuenta también el riesgo de overfitting en los modelos complejos como redes neuronales; y en los modelos menos complejos como regresiones, existe el riesgo de no mejorar la accuracy aun disponiendo de muchos datos, y en ese caso se necesitaría aumentar la dimensionalidad con nuevas variables no fuertemente correladas con las que ya disponemos.

Para evitar los problemas anteriores, modelos más focalizados podían considerarse, como por ejemplo, modelos focalizados por regiones, o por aerolíneas, y así, usar el modelo correspondiente según el vuelo en cuestión. Como en una ruta muchas aerolíneas pueden intervenir, técnicamente sería más óptimo considerar modelos por regiones o países.

Otros modelos que nos hubiera gustado probar son las series temporales. Se trataría, por tanto, de modelos por cada vuelo. Dicho vuelo ha de ser diario. Tendríamos modelos para aquellas rutas más usuales. Aquí la predicción no va a poder ir más allá de un día o dos, pues las series temporales tienden a aumentar la franja de error cuando se pretenden usar para predicciones futuras algo lejanas. En cambio, para alertas o avisos cuando el vuelo este a menos de 48h de salida, ofrecerían un buen modelo a considerar. En esta misma línea de centrarse en los vuelos concretos, la causa de retrasos más frecuente es la que proviene de retrasos acumulados de la misma aeronave, suponiendo que ésta hace vuelos rutinarios, con número de vuelo repetitivo cada día (hipótesis que se da en gran parte de los vuelos), y que existan datos consecutivos; podríamos construir entonces una variable nueva Y_{n-1} que nos diga si la aeronave ha sufrido un retraso en su vuelo el día anterior y usar dicha variable como predictora. El problema aquí es que entonces el modelo servirá únicamente para alertas o vuelos comprados a última hora. Otra variable que podría añadir información relevante en cuanto a la congestión del aeropuerto, es elaborar un rating que cuantifique la cantidad de vuelos que entran y salen del aeropuerto salida en un margen de $\pm 1h$ de la salida programada del vuelo, y así estimar la probabilidad frecuentista de que nuestro vuelo se verá retrasado en salida por congestión en el aeropuerto.

En definitiva, variables clave como retrasos del vuelo anterior y última revisión de la aeronave, no se puede disponer de ellos con mucha antelación, por tanto no se podrían usar para predicciones a 5 días vista, por ejemplo. Una técnica importante a implementar, es la corrección del modelo, comprobar sus aciertos en realidad una vez llegada la fecha, ya que tendremos los datos en el datalake con predicciones y una vez pasada la fecha del vuelo, tendremos la predicción verdadera. Cruzando estos datos evaluamos el modelo. En caso de detectar una baja accuracy en predicciones futuras a muchos días vista, y tras ver qué otros datos pueden influir (y a ser posible incorporar), quizás sería necesario bajar de 5 días a 3 ó 2 días vista.

5. Demo

En este capítulo, mostraremos paso a paso un ejemplo de uso de la aplicación. Cabe destacar que en este prototipo no usamos ningún web service para permitir al usuario hacer uso de nuestras bases de datos sin necesidad de disponer de ellas. Por tanto, para el correcto funcionamiento de la aplicación, se necesita primero de:

- Cuentas de usuario en FlightStats (Cirium) ¹. También cuentas en OpenWeather²
- MongoDB instalado con la base de datos “historicalData” que contiene tres collections: “airportWeather” (alimentada por la aplicación airportWeather.jar 3.1.4), “routeRatings” (alimentada por la aplicación routeRatings.jar 3.1.3) y finalmente “routeStatus” (alimentada por la aplicación routeStatus.jar 3.1.2). Esta última únicamente es necesaria por si se quiere re-entrenar algún modelo con más datos.
- Teniendo lo anterior OK, modificar el fichero `./src/main/resources/confOnTime.xml`³ para indicar las rutas adecuadas. Véase la parte de parametrización de la aplicación 3.2.2
- Las librerías de Python 3 (véase 3.3.4) junto con el Driver Selenium para su navegador (geckodriver ⁴).
- Java 8
- Maven

Una vez tenemos todo lo anterior, compilamos el proyecto con maven usando POM.xml adjunto al proyecto⁵. Tendremos entonces un programa java ejecutable.

A continuación:

1. Iniciar instancia de MongoDB en el puerto por defecto 27017

¹Registrarse aquí <https://developer.flightstats.com/signup>

²Registrarse aquí https://home.openweathermap.org/users/sign_up

³En nueva versión se puede sustituir por un fichero properties, así evitar compilar el programa cada vez que cambien las rutas.

⁴Enlace de descarga <https://github.com/mozilla/geckodriver/releases>

⁵Le dejamos un tutorial para ello <http://www.programandoapasitos.com/2017/07/tutorial-maven-en-eclipse.html>

```

C:\Windows\system32\cmd.exe - mongod.exe
2019-10-24T10:42:20.639+0200 I CONTROL [initandlisten] ** WARNING: This server
is bound to localhost.
2019-10-24T10:42:20.640+0200 I CONTROL [initandlisten] ** Remote syste
ms will be unable to connect to this server.
2019-10-24T10:42:20.641+0200 I CONTROL [initandlisten] ** Start the se
rver with --bind_ip <address> to specify which IP
2019-10-24T10:42:20.641+0200 I CONTROL [initandlisten] ** addresses it
should serve responses from, or with --bind_ip_all to
2019-10-24T10:42:20.642+0200 I CONTROL [initandlisten] ** bind to all
interfaces. If this behavior is desired, start the
2019-10-24T10:42:20.643+0200 I CONTROL [initandlisten] ** server with
--bind_ip 127.0.0.1 to disable this warning.
2019-10-24T10:42:20.644+0200 I CONTROL [initandlisten] Hotfix KB2731284 or late
r update is not installed, will zero-out data files.
2019-10-24T10:42:20.645+0200 I CONTROL [initandlisten]
2019-10-24T10:42:21.210+0200 W FTDC [initandlisten] Failed to initialize Per
formance Counters for FTDC: WindowsPdhError: PdhExpandCounterPathW failed with '
El objeto especificado no se encontró en el equipo.' for counter '\Memory\Availa
ble Bytes'
2019-10-24T10:42:21.212+0200 I FTDC [initandlisten] Initializing full-time d
iagnostic data capture with directory 'C:/data/db/diagnostic.data'
2019-10-24T10:42:21.215+0200 I NETWORK [initandlisten] waiting for connections
on port 27017

```

Figura 5.1: Sesión MongoDB

2. Ejecutar ontime.jar con doble clic
3. Completar los parámetros que deseemos

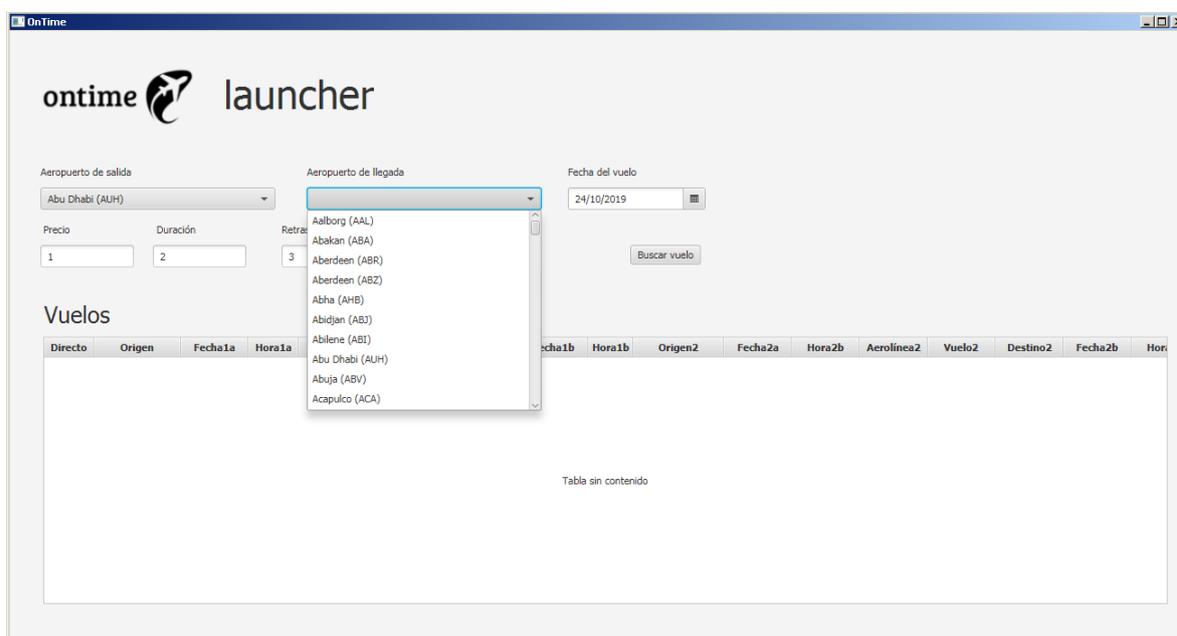


Figura 5.2: Interfaz OnTime

4. Clicar sobre el boton “Buscar vuelos”
5. Si los parámetros son correctos, encontrará los vuelos y los mostrará en la tabla inferior. En caso de no existir vuelos, bien porqué no los hay para su ruta y fecha, o bien porque

no está aún incluida su ruta en el prototipo, recibirá el mensaje correspondiente.

6. Escoge su vuelo y clicas sobre el link de la columna HREF, se redirigirá a realizar la compra de su billete.

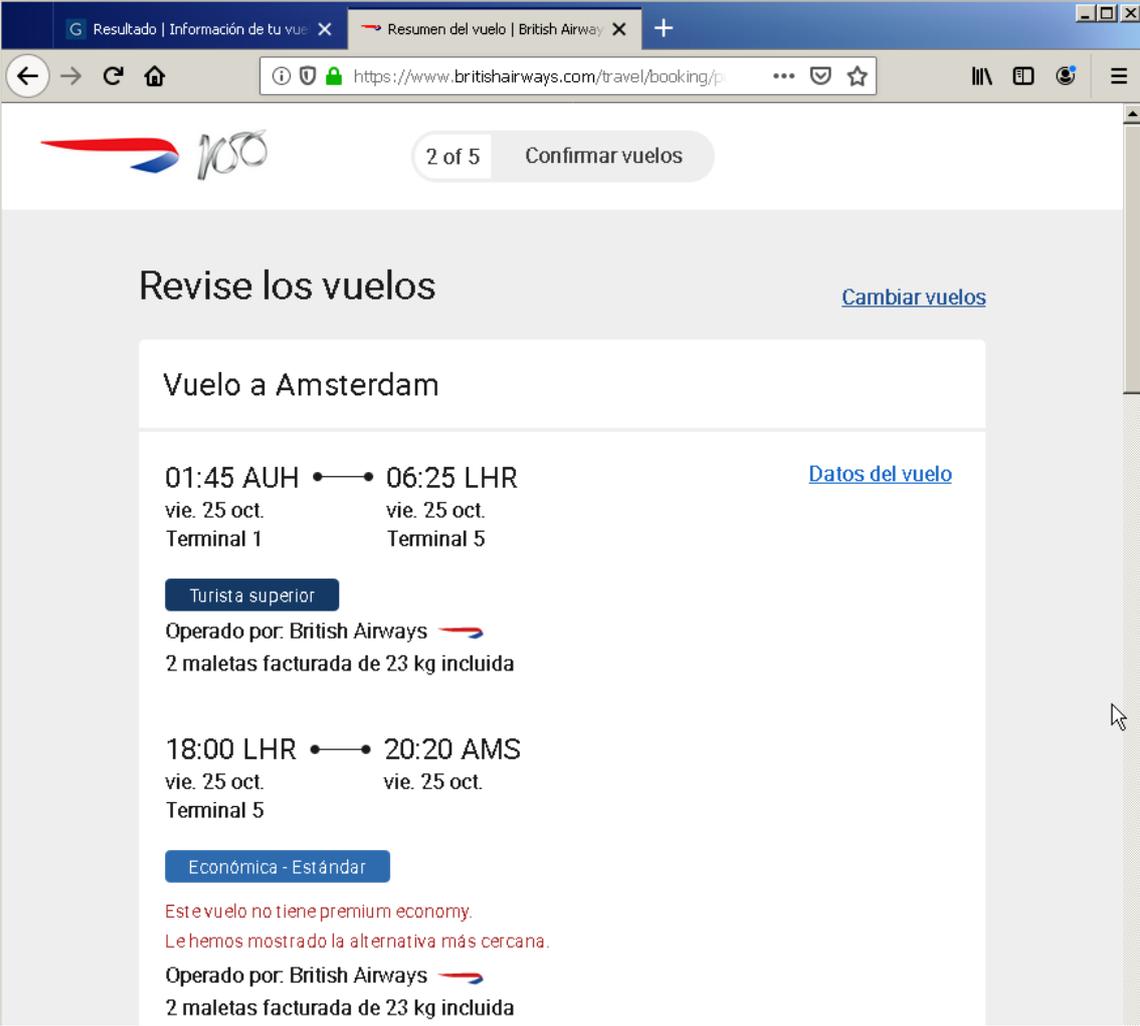
The screenshot shows the 'OnTime launcher' interface. At the top, there are search filters for 'Aeropuerto de salida' (Abu Dhabi (AUH)), 'Aeropuerto de llegada' (Amsterdam (AMS)), and 'Fecha del vuelo' (25/10/2019). Below these are input fields for 'Precio' (1), 'Duración' (2), and 'Retraso' (3), along with a 'Buscar vuelo' button.

The main section is titled 'Vuelos' and contains a table with the following columns: Vuelo, Destino, Fecha1b, Hora1b, Origen2, Fecha2a, Hora2b, Aerolínea2, Vuelo2, Destino2, Fecha2b, Hora2b, Precio, Duración, HREF, and Delayed. The HREF column contains links for each flight entry.

Vuelo	Destino	Fecha1b	Hora1b	Origen2	Fecha2a	Hora2b	Aerolínea2	Vuelo2	Destino2	Fecha2b	Hora2b	Precio	Duración	HREF	Delayed
9655	FRA	2019-10-25	06:45	FRA	2019-10-25	08:10	LH	986	AMS	2019-10-25	09:20	695	9h 10m	https://w...	30.0
9661	MUC	2019-10-25	06:55	MUC	2019-10-25	08:50	LH	2302	AMS	2019-10-25	10:20	730	9h 45m	https://w...	30.0
9655	FRA	2019-10-25	06:45	FRA	2019-10-25	09:05	LH	988	AMS	2019-10-25	10:15	695	10h 05m	https://w...	30.0
72	LHR	2019-10-25	06:25	LHR	2019-10-25	08:20	BA	430	AMS	2019-10-25	10:40	948	10h 55m	https://w...	30.0
9655	FRA	2019-10-25	06:45	FRA	2019-10-25	12:40	LH	992	AMS	2019-10-25	13:50	695	13h 40m	https://w...	30.0
85	FCO	2019-10-25	06:20	FCO	2019-10-25	08:45	AZ	108	AMS	2019-10-25	11:15	1165	11h 00m	https://w...	30.0
9661	MUC	2019-10-25	06:55	MUC	2019-10-25	13:45	LH	2304	AMS	2019-10-25	15:15	730	14h 40m	https://w...	30.0
72	LHR	2019-10-25	06:25	LHR	2019-10-25	11:45	BA	434	AMS	2019-10-25	14:05	816	14h 20m	https://w...	30.0
72	LHR	2019-10-25	06:25	LHR	2019-10-25	12:35	BA	438	AMS	2019-10-25	14:55	816	15h 10m	https://w...	30.0
9661	MUC	2019-10-25	06:55	MUC	2019-10-25	15:15	LH	2306	AMS	2019-10-25	16:45	730	16h 10m	https://w...	30.0

Figura 5.3: Interfaz OnTime

7. Si lo desea, puede reservar!



The screenshot shows a web browser window with two tabs: 'Resultado | Información de tu vuelo' and 'Resumen del vuelo | British Airways'. The address bar shows the URL 'https://www.britishairways.com/travel/booking/p'. The page header includes the British Airways logo, a '2 of 5' indicator, and a 'Confirmar vuelos' button. The main content area is titled 'Revise los vuelos' with a 'Cambiar vuelos' link. It displays two flight segments for a round trip to Amsterdam on Friday, October 25th. The first segment is from AUH to LHR (Terminal 1 to Terminal 5) at 01:45 to 06:25, with 'Turista superior' class. The second segment is from LHR to AMS (Terminal 5) at 18:00 to 20:20, with 'Económica - Estándar' class. Both flights are operated by British Airways and include 2 checked bags of 23 kg each. A note states that the second flight does not have premium economy and that the closest alternative has been shown.

Resultado | Información de tu vuelo × Resumen del vuelo | British Airways × +

https://www.britishairways.com/travel/booking/p

2 of 5 Confirmar vuelos

Revise los vuelos [Cambiar vuelos](#)

Vuelo a Amsterdam

01:45 AUH — **06:25 LHR**
vie. 25 oct. Terminal 1 vie. 25 oct. Terminal 5 [Datos del vuelo](#)

Turista superior
Operado por: British Airways
2 maletas facturada de 23 kg incluida

18:00 LHR — **20:20 AMS**
vie. 25 oct. Terminal 5 vie. 25 oct.

Económica - Estándar
Este vuelo no tiene premium economy.
Le hemos mostrado la alternativa más cercana.
Operado por: British Airways
2 maletas facturada de 23 kg incluida

Figura 5.4: Página proveedor vuelo, puede realizar su compra

8. Realiza otra búsqueda, cambiando parámetros y clicando en buscar vuelo.

6. Conclusiones

Una de las primeras cosas que hemos podido constatar en este trabajo, es que la importancia de la arquitectura y diseño en las bases de datos de documentos es crucial de cara a que la aplicación sea funcional. No únicamente el diseño de los documentos según las queries más usadas, sino también los índices y las claves.

Otra de las cosas que hemos podido practicar en este proyecto, es que según qué problema en el análisis de datos, requiere de un mayor o menor volumen. En nuestro caso, y como se comentó en la sección de conclusiones del capítulo Data Analytics 4.5, las variables tienen muchos levels, y por tanto para tenerlos todos representados, se necesitan muchos datos. Lo anterior no es un punto negativo, si no que nos muestra que técnicas del Big Data son de utilidad en Data Analysis para entrenar modelos sobre millones de registros que las librerías tradicionales no serían capaces de tratar. En la misma línea, podemos concluir que conocer las variables más relacionadas con la variable respuesta es crucial, y disponer de diversas variables así supondría un empuje al modelo. Por otra parte, variables que en el día a día sabemos que afectan a los retrasos no han sido contempladas. Es difícil medir huelgas¹, declaraciones políticas², etc. que claramente afectan a los retrasos. En todo caso, siempre está la posibilidad de considerar incorporar dichas medidas usando técnicas de análisis del lenguaje y streaming sobre redes sociales.

Lo anterior nos sugiere dos importantes conclusiones adicionales: una es que los datos, sin duda, generan negocio, lo que reafirma una vez más el auge de Big Data y Data Analysis; y otra, es que los modelos se han de complementar con otros para enriquecerse. En nuestro caso, no se puede esperar que las predicciones vayan muy lejos en el tiempo, y los modelos se han de re-entrenar con frecuencia, así como las predicciones deben ser 5 días máximo.

El método de Ensemble learning, GBTC, ha mostrado ser más útil para el tipo de problemas tratado aquí, con accuracy en torno a 69%. Esta accuracy podría mejorar con datos más heterogéneos, pero para este tipo de problemas, aparte de buscar una mejor accuracy involucrando diferentes modelos, se ha de buscar también confianza en el modelo. Para ello, habría que reducir el margen de días a los que se aplica el algoritmo, así como re-entrenar con mayor frecuencia los modelos (lo que conlleva investigar aquellos que son capaces de re-entrenar sobre lo ya aprendido).

Se han podido practicar pruebas de entrenamiento de modelos sobre Cluster. Hemos notado que hay que tener especial cuidado con el shuffling y los algoritmos a nivel de código, prestar atención al hecho de que sea necesario mandar la información unificada a un máster o slave en concreto. Teniendo en cuenta estas salvedades técnicas y de “lógica” del paralelismo,

¹Ejemplo Huelgas Cataluña Octubre 2019.

²Ejemplo, declaraciones de EEUU hacia Turquía, Octubre 2019.

se observa la utilidad de dicha infraestructura en Data Analysis. Aunque no hemos notado una mejora sustancial respecto de las pruebas en local, consideramos que se debe a que los datos usados no son exageradamente grandes como para captar una diferencia significativa.

Bibliografía

- Alice, S., Jorge, S., Diego, C., y Eduardo, O. (2017). A review on flight delay prediction.
- Anish, M. K., y Aera, K. L. (2017). Predictive modeling of aircraft flight delay. , 5(10), 485–491.
- Charles, N. G., y Michael, O. B. (2013). Stochastic optimization models for ground delay program planning with equity–efficiency tradeoffs. , 33, 196–202.
- Cirium. (s.f.-a). *The airport response*. Descargado de <https://developer.flightstats.com/api-docs/airports/v1/airportresponse>
- Cirium. (s.f.-b). *Flight status response element*. Descargado de <https://developer.flightstats.com/api-docs/flightstatus/v2/flightstatusresponse>
- Cirium. (s.f.-c). *Ratings response element*. Descargado de <https://developer.flightstats.com/api-docs/ratings/v1/ratingsresponse>
- Martina, Z., Martin, P., y Radek, S. (2017). Factors influencing flight delays of a european airline. , 65(5), 1799–1807.
- openweather. (s.f.). *5 day weather forecast*. Descargado de <https://openweathermap.org/forecast5>
- Stefan, B., y Christian, U. (2019). Bad weather and flight delays: The impact of sudden and slow onset weather events. , 18(5), 10–26.

A. Anexo I - Módulo Data Analytics

Datos estáticos

Ejemplo documento Airports API Cirium (s.f.-a)

```
1 {
2   "airports": [
3     {
4       "fs": "BCN",
5       "iata": "BCN",
6       "icao": "LEBL",
7       "faa": "",
8       "name": "Barcelona-El Prat Airport",
9       "city": "Barcelona",
10      "cityCode": "BCN",
11      "stateCode": "SP",
12      "countryCode": "ES",
13      "countryName": "Spain and Canary Islands",
14      "regionName": "Europe",
15      "timeZoneRegionName": "Europe/Madrid",
16      "weatherZone": "",
17      "localTime": "2019-07-17T11:14:50.061401",
18      "utcOffsetHours": 2,
19      "latitude": 41.303027,
20      "longitude": 2.07593,
21      "elevationFeet": 13,
22      "classification": 1,
23      "active": true,
24      "weatherUrl": "https://api.flightstats.com/flex/weather/rest/↔
↔ v1/json/all/BCN?codeType=fs",
25      "delayIndexUrl": "https://api.flightstats.com/flex/delayindex↔
↔ /rest/v1/json/airports/BCN?codeType=fs"
26    }
27  ]
}
```

Muestra fichero aeropuertos+coordenadas:

```
1 BAV;40.563442;109.998495
2 BAX;53.361086;83.547643
3 BBI;20.252853;85.817385
4 BCD;10.644815;122.933593
5 BCN;41.303027;2.07593
6 BDA;32.35994;-64.701148
7 BDJ;-3.43804;114.754253
```

Muestra fichero rutas:

```
1 BAQ;BOG
2 BAQ;MIA
3 BAX;DME
4 BBI;DEL
5 BCD;MNL
6 BCN;AAL
7 BCN;ACE
8 BCN;AGP
```

RouteStatus

Fichero properties para el programa routeStatus.jar

```
1 appId=b69e93cd
2 appKey=5fea1fc894f717af54f490df54c3ae2f
3 departureAirport=BCN
4 arrivalAirport=
5 year=-1
6 month=7
7 day=25
8 hourOfDay=0
9 numHours=24
10 reformat=Y
11 untilYesterday=
12 data_path=C:/Users/AMIGO/eclipse-workspace/data/rutas_final
13 database=historicalData
14 collection=routeStatus_v2
```

Muestra json routeStatus:

```
1 {
2   "_id": "AAEALG20197220",
3   "departureAirport": {
4     "requestedCode": "AAE",
5     "fsCode": "AAE"
6   },
7   "arrivalAirport": {
8     "requestedCode": "ALG",
9     "fsCode": "ALG"
```

```
10 },
11 "date":{
12   "year":"2019",
13   "month":"7",
14   "day":"22",
15   "interpreted":"2019-07-22"
16 },
17 "flightStatuses":[
18   {
19     "flightId":1007810218,
20     "carrierFsCode":"AH",
21     "flightNumber":"6007",
22     "departureAirportFsCode":"AAE",
23     "arrivalAirportFsCode":"ALG",
24     "departureDate":{
25       "dateLocal":"2019-07-22T07:15:00.000",
26       "dateUtc":"2019-07-22T06:15:00.000Z"
27     },
28     "arrivalDate":{
29       "dateLocal":"2019-07-22T08:15:00.000",
30       "dateUtc":"2019-07-22T07:15:00.000Z"
31     },
32     "status":"L",
33     "schedule":{
34       "flightType":"J",
35       "serviceClasses":"FY",
36       "restrictions":""
37     },
38     "delays":{
39       "departureGateDelayMinutes":20,
40       "arrivalGateDelayMinutes":15
41     },
42     "codeshares":null,
43     "flightDurations":{
44       "scheduledBlockMinutes":60,
45       "blockMinutes":55,
46       "airMinutes":35,
47       "taxiOutMinutes":10,
48       "taxiInMinutes":10
49     },
50     "airportResources":{
51       "arrivalTerminal":"D"
52     }
53   }
54 ]
55 }
```

RouteRatings

Fichero properties para el programa routeRatings.jar

```
1 appId=6d5fed86
2 appKey=0572c89419c734f13b5f9bc6fec6372a
3 departureAirport=BCN
4 arrivalAirport=
5 reformat=Y
6 database=historicalData
7 collection=routeRatings_v2
8 reformat2=Y
9 data_path=./rutas_final
```

Muestra json routeRatings

```
1 {
2   "_id": "BCN-NDR-VY-7378",
3   "airlineFsCode": "VY",
4   "flightNumber": "7378",
5   "departureAirportFsCode": "BCN",
6   "arrivalAirportFsCode": "NDR",
7   "codeshares": 0,
8   "directs": 0,
9   "observations": 10,
10  "ontime": 2,
11  "late15": 3,
12  "late30": 2,
13  "late45": 3,
14  "cancelled": 0,
15  "diverted": 0,
16  "ontimePercent": 0.2,
17  "delayObservations": 9,
18  "delayMean": 40.0,
19  "delayStandardDeviation": 27.893149298309392,
20  "delayMin": 4,
21  "delayMax": 90,
22  "allOntimeCumulative": 0.0,
23  "allOntimeStars": 0.0,
24  "allDelayCumulative": 0.3085,
25  "allDelayStars": 1.5,
26  "allStars": 0.75
27 }
```

AirportWeather

Fichero properties para el programa airportWeather.jar

```
1 appKey=be1705e55796eeb65d257be8754a31d1
2 latitude=41.303027
3 longitude=
4 airport=BCN
5 data_path=./aeropuertos_final+coordenadas
6 database=historicalData
7 collection=airportWeather_v3
```

Muestra json airportWeather

```
1 {
2   "_id": "BCN-2019-9-17-17",
3   "city": {
4     "id": 6544426,
5     "name": "El Mas Blau",
6     "coord": {
7       "lat": 41.3177,
8       "lon": 2.0737
9     },
10    "country": "ES",
11    "timezone": 7200
12  },
13  "list": [
14    {
15      "dt": 1569164400,
16      "main": {
17        "temp": 301.6,
18        "temp_min": 301.6,
19        "temp_max": 301.6,
20        "pressure": 1014.95,
21        "humidity": 44,
22        "temp_kf": 0
23      },
24      "weather": [
25        {
26          "main": "Rain"
27        }
28      ],
29      "wind": {
30        "speed": 4.13,
31        "deg": 149.148
32      },
33      "rain": {
```

```
34         "3h":0.562
35     },
36     "dt_txt":"2019-09-22 15:00:00",
37     "dt_local":1569171600
38 }
39 ],
40 "airport":"BCN",
41 "dt_0":1568750400,
42 "dt_1":1569171600,
43 "dt_0_utc":1568743200,
44 "dt_1_utc":1569164400
45 }
```

Dataset

Fichero properties para e programa dataset.jar

```
1timeZone=
2data_path=./dataset.csv
3database=historicalData
4collection_routeStatus=routeStatus
5collection_routeRatings=routeRatings
6collection_airportWeather=airportWeather
```

Model

Fichero properties para e programa Model.jar

```
1# Model with weather data N Y
2weather=Y
3
4# choose the model
5# -----
6# RF for Random forests
7# RF_CV for Random Forests witch CrossValidation
8# GBTC for Gradient-boosted tree classifier
9# LinearSVM sor Support Vector Machine
10# OneVsRestLR one versus rest with logistic regression
11# OneVsRestSVM one versus rest with Support Vector Machine
12# LR for logistic regression
13# NB for Naive Bayes
14# Multilayer perceptron classifier feedforward neural network
15model=
16
17# binary classification vs multiclass-classification
```

```
18 multiclass=N
19
20 # Model path to save .mod
21 # we save Indexes in ./Index
22 model_path=hdfs://master:27000/user/bdma47/
23
24 # Data path
25 data_path=hdfs://master:27000/user/bdma47/dataset_3.csv
```

B. Anexo II - Módulo búsqueda de vuelos

DataLake

Muestra fichero aeropuertos:

```
1Banjarmasin (BDJ)
2Banjul (BJL)
3Baotou (BAV)
4Barcelona (BCN)
5Bari (BRI)
6Barnaul (BAX)
7Barrancabermeja (EJA)
8Barranquilla (BAQ)
```

Rutas - API Connections

La API Connections nos da vuelos entre dos aeropuertos incluyendo escalas. Los vuelos directos también los proporciona.

Este es un ejemplo de llamada, para buscar conexiones de Barcelona a Moscú para el día 19 de octubre de 2019. https://api.flightstats.com/flex/connections/rest/v2/json/firstflightin/BCN/to/DME/arriving_before/2019/10/19/23/59?includeCodeshares=true&payloadType=passenger&maxResults=250&includeMultipleCarriers=true&numHours=24&includeSurface=false&maxConnections=1

El resultado es un json cuyo aspecto compacto es este:

```
1{
2  "request": {
3  "connections": [{
4  "appendix": {
5    "airlines": [{
6    "airports": [{
7    "equipment": [{
8  }
9}
```

El primer atributo, “request” es la query efectuada devuelta de manera tabulada. El atributo “appendix” contiene a su vez tres más que podrían ser interesantes, pero no los hacemos servir en este prototipo. En “airlines” contiene la lista de aerolíneas que aparecen en el atributo “connections”, que es el que contiene las rutas del vuelo. Lo mismo para “airports” (qué aeropuertos atraviesan las rutas devueltas) y “equipments”

(tipo de avión, es un jet o no, etc.)

El atributo que nos resulta más interesante, como se ha comentado, es el atributo “connections”, que pasamos a describir a continuación, de cara a entender la formación de las rutas cuando un usuario pide comprar un vuelo.

Este es un ejemplo(una muestra no completa del json respuesta) de un vuelo que devuelve la llamada a la API anterior:

```
1 {
2   "request": {
3     "connections": [{
4       "elapsedTime": 265,
5       "score": 75,
6       "scheduledFlight": [{
7         "carrierFsCode": "U6",
8         "flightNumber": "846",
9         "departureAirportFsCode": "BCN",
10        "arrivalAirportFsCode": "DME",
11        "stops": 0,
12        "departureTime": "2019-10-25T14:45:00.000",
13        "arrivalTime": "2019-10-25T20:10:00.000",
14        "flightEquipmentIataCode": "321",
15        "isCodeshare": false,
16        "isWetlease": false,
17        "serviceType": "J",
18        "trafficRestrictions": [],
19        "codeshare": [{
20          "carrierFsCode": "9U",
21          "serviceType": "J"}],
22        "elapsedTime": 205
23      }]
24    },
25    {
26      "elapsedTime": 340,
```

Connections por tanto es un array document que contiene documentos, llamemoste connection Document. Dentro de cada connection existe un array document “schedulesFlight”, con documentos “Schedule”.

Los documentos “Schedule” representan vuelos, y schedulesFlight podemos verlo como rutas. Si el vuelo es directo tendremos “scheduledFlight” con un único documento embebido, con información del vuelo, en caso de vuelo con escala, el array document “scheduledFlight” dispondrá de documentos consecutivos, el primero para la primera parte del vuelo, y el segundo para el vuelo posterior a la escala.

En un documento shedule, podemos encontrar un sub array document llamado “Codes-

hares”, aquí se encontrarán los codeshare que el vuelo comparte con otras aerolíneas. Esto provoca que ciertos datos de ese vuelo sean modificados en la práctica, como la aerolínea (“carrier”) o el número de vuelo¹.

Es importante tener en cuenta estos codeshares, ya que, a la hora de comprar un vuelo, no es lo mismo que te lo gestione “IB” (Iberia) que “VY” (Vueling), ya que se pagarían a compañías diferentes. Además, en el prototipo usamos scraping para obtener los precios, pues se da el caso de que un vuelo puede estar a la venta con un código u otro, según qué compañía de las que gestionan ese vuelo, aun le quedan billetes.

En la práctica, el sistema considera los nodos codeshare como un vuelo alternativo más, que contiene toda la información del nodo “Schedule” en el que se encuentra, sustituyéndolos donde sea necesario por los datos propios del nodo “codeshare” (número de vuelo, carrier, o ambos).

El algoritmo de búsqueda de rutas tiene en cuenta todo lo comentado para extraer todas las posibilidades existentes en el response de la API, de cara a servir la petición del usuario.

¹La versión de pago siempre ofrece el atributo flightNumber para el codeshare asociado, cosa que la versión gratuita no siempre alimenta este atributo. Técnicamente esto provoca complicaciones que en realidad no tendrían porque existir.

C. Anexo III - Sobre análisis

El objetivo de este capítulo es hacer un análisis descriptivo de los datos, aeropuertos con mayor retraso, aerolíneas con mayor retraso, momentos del día con mayor retraso, regresión lineal entre variables y visualización.

Cantidad de Datos

A nivel de cantidad de vuelos que disponemos a la hora de entrenar modelos son :

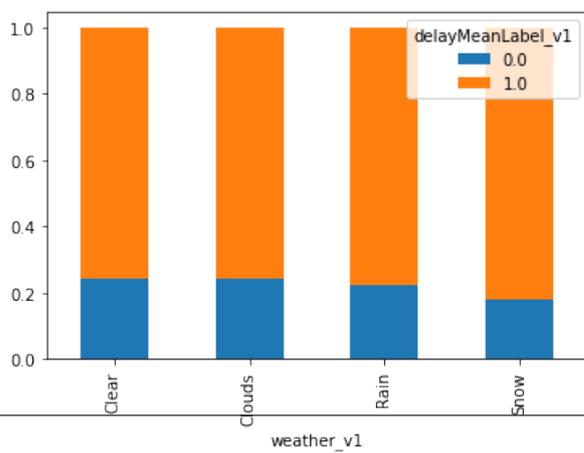
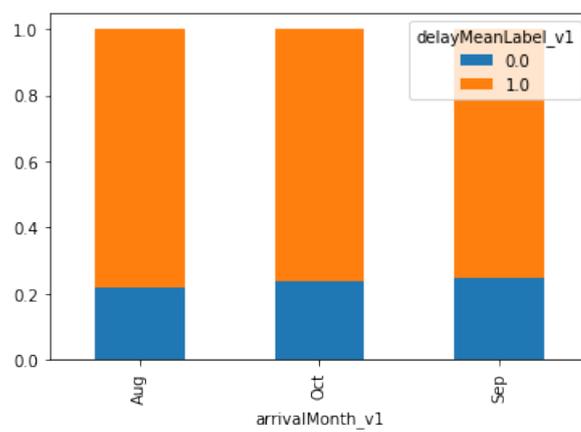
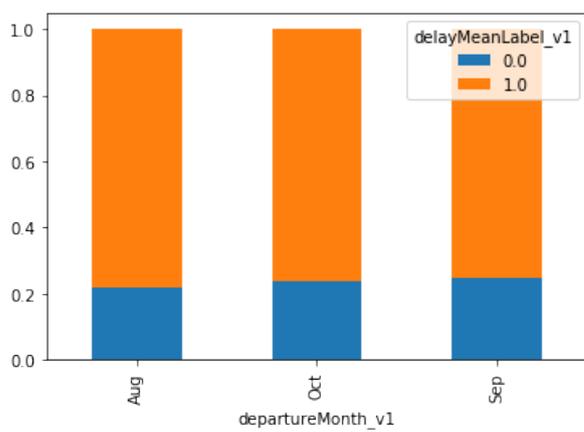
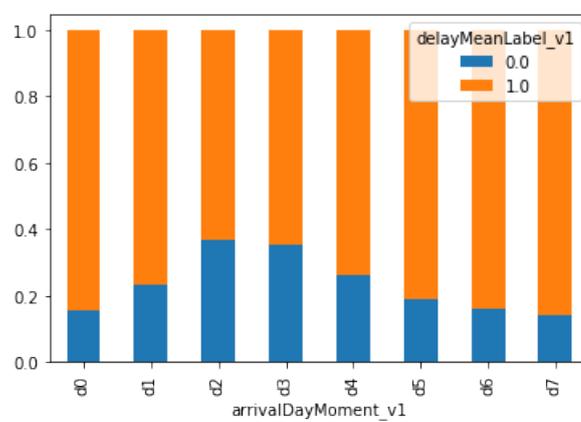
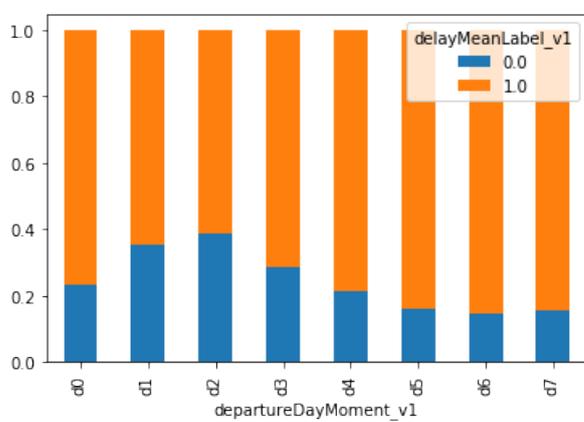
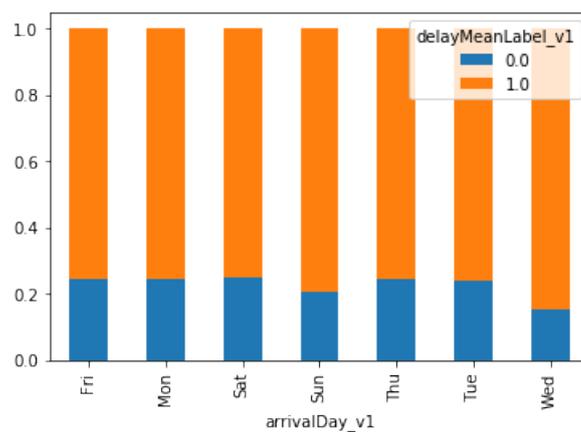
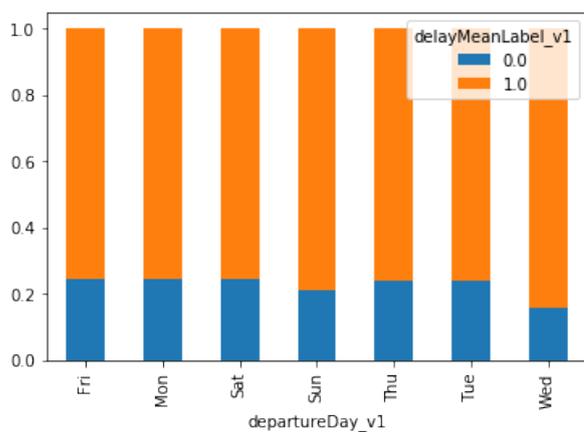
departureMonth_v1	Aug	Oct	Sep	tot
departureDay_v1				
Fri	0	0	51063	51063
Mon	2	0	50067	50069
Sat	45568	0	42541	88109
Sun	48111	0	447	48558
Thu	0	0	48504	48504
Tue	0	45893	43	45936
Wed	0	32	333	365
Tot	93681	45925	192998	332604

Tabla C.1: Tabla de contingencia entre departureDay vs departureMonth

Como vemos faltan registros representativos de varias combinaciones lo cual puede dar cierto sesgo a los modelos que podamos hacer ya que los datos están sesgados por los días en los que se cogieron.

Histogramas, en qué momentos hay más retrasos

```
1 tcs=[]
2 for i in range(0,len(to_keep_categorical)):
3     tc=pd.crosstab(data[to_keep_categorical[i]],data['delayMeanLabel_v1'])
4     tcs.append(tc)
5     tc.div(tc.sum(axis=1),axis=0).plot(kind="bar",stacked=True)
```



En las gráficas relativas al momento del día acumulándose mas retrasos sobre finales del día que durante la mañana de 6:00 a 9:00 como horas con menos retrasos, para lo que se refiere a días de la semana el día con algo menos de retrasos es el Miércoles con poca diferencia. En lo referido al tiempo es una variable bastante accesoria, solo en casos de nieve parece haber mas retrasos.

Qué aeropuertos y qué aerolíneas sufren más retrasos, y menos ?

Columna1	mean	Std	delayMean_v1
departureAi			
SZX	62,446	34,899	1,791
ORD	55,905	32,094	1,742
HGH	54,466	31,206	1,747
EWR	54,435	36,277	15,007
PEK	53,558	31,583	1,696
...
HKT	16,175	14,924	1,085
CNX	16,097	18,495	8,720
CGK	16,002	14,906	1,074
FUK	15,116	9,567	1,581
SUB	14,425	13,092	1,103

Los aeropuertos que registran mayor retraso en media(salida) son el aeropuerto de Hangzhou (China), el aeropuerto internacional O'Hare de Chicago (USA) y el aeropuerto internacional de Hangzhou Xiaoshan (China), en cuanto a los top-3 con menor retrasos en media son: aeropuerto internacional de Surabaya (Indonesia), aeropuerto de Fukuoka (Japon) y aeropuerto internacional de Jakarta Soekarno-Hatta (Indonesia).

Sin embargo en la mayoría de casos la desviación estándar es considerable comparado con la media, con lo cual los vuelos se distribuyen ampliamente.

Columna1	mean	Std	delayMean_v1
carrierFsCoc			
C5	65,058	38,887	1,674
ZW	61,291	30,245	2,027
B6	60,439	30,895	1,956
EV	52,139	34,880	1,495
AX	50,831	32,875	1,547
...
PG	14,780	11,366	1,301
JT*	14,720	19,477	0,756
CL	14,249	74,005	1,926
GA	14,022	14,110	0,994
ID*	13,870	11,710	1,185

Con lo cual el top-3 de las “peores” aerolíneas en cuanto a retrasos tenemos C5 (COMMUT Aire), ZW (Air Zimbabwe) y B6 (JetBlue). Las “mejores” compañías con menos retrasos tenemos ID* (Batik Air), GA (Garuda Indonesia) y CL (Lufthansa CityLine).

Notese que las dos primeras “mejores” aerolíneas son de Indonesia, donde hemos obtenido aeropuertos con menos retrasos, y las peores aerolíneas, una de ellas corresponde a USA donde aparece en el top-3 de aeropuertos con más retrasos.

Nota: la desviación típica es grande comparada con la media, con lo cual el orden no es estricto.
