Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Entendiendo los flujos migratorios de la Unión Europea con Big Data

Understanding the migration flows of the European Union with Big Data

Alberto Sarabia Suárez

La Laguna, 10 de septiembre de 2020

D. **Marcos Colebrook Santamaría**, con N.I.F. 43.787.808-V profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **Jesús M. Jorge Santiso**, con N.I.F. 42.097.398-S profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas, como cotutor

C E R T I F I C A N

Que la presente memoria titulada:

"*Entendiendo los flujos migratorios de la Unión Europea con Big Data*"

ha sido realizada bajo su dirección por D. **Alberto Sarabia Suárez**, con N.I.F. 51.152.079-X.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 10 de septiembre de 2020.

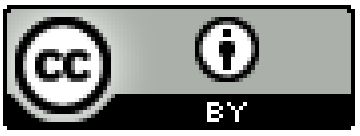
Agradecimientos

Agradecer a mi familia por el apoyo que me han proporcionado a lo largo de mis estudios en Ing Informática por su forma única de lograr entender y relajarme en las largas horas que pasaba en la biblioteca y por estar allí conmigo en cada mal examen para apoyarme.

Agradecer a mis amigos que he conocido en los años que he estado cursando el Grado de Ingeniería Informática por las chuletadas, por las quedadas, por los largos días en la biblioteca estudiando para los exámenes o cuando desarrollamos las prácticas, por explicarme lo que no entendía y por tenerme paciencia, gracias.

Agradecer al profesorado de la Universidad de La Laguna que presta docencia en el Grado de Ingeniería en Informática que a pesar de no ser de los estudiantes más entendido o dedicado a los estudios siempre han estado allí para cualquier duda que surgiera y para ayudar en el desarrollo tanto de las prácticas como de la preparación de los exámenes finales, siendo severos cuando debían serlo por nuestro bien y para que, al salir al mundo laboral, estuviéramos lo mejor preparados para todos los retos que nos irán surgiendo en esta vida laboral y en el entorno tan cambiante como tenemos hoy en dia.

Licencia



Resumen

A medida que el ser humano ha sido capaz de desarrollar la tecnología, se ha desarrollado también la cantidad de información que se genera en el mundo. No solo las personas generan información, sino que también las mismas aplicaciones y equipos informáticos. Del deseo de sacar valor añadido de la información generada por las empresas e instituciones públicas nació el análisis de datos, llegando a nuestros días en donde la cantidad de información es tan ingentemente grande que surge el *Big Data* como una forma nueva de entender el flujo de información y la forma de explotarlo de forma eficiente.

Los recursos en nuestro planeta son limitados, así como lo son los presupuestos de la Unión Europea (UE). Solo el gasto en política de migración exterior de la UE en los países de la vecindad oriental y del mediterráneo meridional hasta 2014 sobre veintitrés de los proyectos contratados ha sido de 89 millones de euros de un total de 742 millones.

He decidido hacer este proyecto para intentar arrojar un poco de luz sobre los flujos migratorios utilizando el Big Data para ello y, así, intentar conseguir un mejor rendimiento de los pocos recursos que la UE invierte en inmigracion.

Este informe, al utilizar Big Data, buscará las fuentes de datos más diversas y de diferentes orígenes siempre sean datos oficiales o de instituciones cuyo método de recogida de datos pueda avalar que los datos analizados son correctos. Esto es así porque deseamos tener un *dataset* lo más completo y diverso posible.

Finalmente, se ha desarrollado un ecosistema capaz de conectar con un sistema gestor de base de datos (MongoDB), utilizar una herramienta de procesamiento en paralelo de almacenamiento distribuido (Hadoop) y que es capaz de realizar informes programables con una herramienta de visualización de datos (Pentaho).

Palabras clave: MongoDB, Hadoop, YARN, HDFS, Pentaho, Clustering, k-means

Abstract

As human beings have been able to develop technology, the amount of information that is generated in the world has also been developed. Not only people generate information but also the same applications and computer equipment as servers. From the desire to extract added value from the information generated by companies and public institutions, data analysis was born. Nowadays, where the amount of information is so massively large, Big Data is born as a new way of understanding the flow of information and the way of exploiting it efficiently.

The resources on our planet are limited as are the budgets of the European Union (EU). Only the Spending on the EU's foreign migration policy in the countries of the eastern neighborhood and the southern Mediterranean until 2014 in twenty-three projects of the contracted projects has been 89 million euros out of a total of 742 million.

I have decided to do this project to try to shed some light on migratory flows using Big Data, and thus, try to get a better return on the few resources that the European Union invests in immigration.

This report, when using Big Data, will search the most diverse data sources and from different origins whenever official data or from institutions whose data collection method or reputation can guarantee that the data analyzed is right. This is because we want to have as complete and diverse a dataset as possible.

Finally, an ecosystem has been developed capable of connecting with a database management system (MongoDB), using a distributed storage parallel processing tool (Hadoop) and capable of producing programmable reports with a data visualization tool (Pentaho).

Keywords: MongoDB, Hadoop, YARN, HDFS, Pentaho, Clustering, k-means

Índice General

Introducción **1**
¿Por qué el Big Data? 1
Objetivo General 2
Objetivos Específicos 2
Requisitos 2
Aplicación real del campo de estudio. 3
Estado del arte 4
A migration equilibrium model with uncertain data and movement costs 4
Human migration networks and policy interventions: bringing population distributions in line with system optimization 4
Uso de Big Data para entender las migraciones 4
El equipo de Facebook Data Science 5
E-GEOS 5

Tecnologías usadas **6**
¿Qué es el Big Data? 6
Tipos de herramientas utilizadas 7
Herramientas de gestión de datos 7
Sistemas Relacionales (SQL) 7
Sistemas No Relacionales (NoSQL) 8
Herramientas y técnicas de Análisis 8
Técnicas de análisis 8
Herramientas de visualización de datos 10
Técnicas de desarrollo utilizadas 10
¿Qué es el Test Driven Development (TDD)? 10
¿Por qué usamos Test Driven Development (TDD)? 10
¿Qué es el Behavior Driven Development (BDD)? 11
¿Por qué usamos Behavior Driven Development (BDD) ? 11
Design Patterns 12
¿Qué son los Desing Patterns o Patrones de Diseño? 12
Patrones usados en este TFG 12
¿Por qué usamos los Patrones de Diseño? 14
Java 14

¿Qué es Java? 14
¿Por qué usamos Java ? 14
Software utilizado 15
Maven 15
¿Por qué utilizamos Maven? 16
MongoDB 16
¿Por qué utilizamos MongoDB? 17
¿Qué es el software de agregación? 17
¿Por qué utilizamos el aggregation framework? 18
Hadoop 18
¿Por qué usamos Hadoop? 20
¿Qué es YARN (Yet Another Resource Negotiator)? 20
¿Por qué usamos YARN? 22
Pentaho 22
¿Por qué usamos Pentaho? 22
K-means 22
Desarrollo del ecosistema y flujo de trabajo 24
Clases desarrolladas 24
Flujo de trabajo Ecosistema Big Data. 28
Tratamiento de Datos 34
Conclusiones y líneas futuras 36
Conclusions and future work 38
Presupuesto 40
Costes de Hardware 40
Costes de Recursos Humanos 40
Costes Totales 41
Bibliografía 42

Índice de figuras

Figura 2.1. Ejemplo de clustering 8
Figura 2.2. Ejemplo de árbol de decisión 9
Figura 2.3. Ejemplo de Deep Learning 10
Figura 2.4. Ejemplo de TDD 11
Figura 2.5. Ejemplo BDD 12
Figura 2.6. Ejemplo de Singleton pattern 13
Figura 2.7. Ejemplo de Adapter pattern 13
Figura 2.8. Ejemplo de MCV architecture 14
Figura 2.9. Ejemplo de pom.xml 16
Figura 2.10. MongoDB 17
Figura 2.11. Ejemplo de fichero JSON 17
Figura 2.12. Ejemplo de query en aggregation framework 17
Figura 2.13. Hadoop Ecosystem 18
Figura 2.14. Ejemplo de HDFS architecture 19
Figura 2.15. Ejemplo de MapReduce 20
Figura 2.16. Ejemplo de arquitectura en YARN 21
Figura 2.17. Logo de Pentaho 22
Figura 2.18. Método del codo 23
Figura 2.19. Ejemplo de k-means 23
Figura 3.1. Ecosistema Big Data en UML 25
Figura 3.2. Ejemplo de config file en MongoDB 31
Figura 3.3. Ejemplo config file de Hadoop 32
Figura 3.4. Ejemplo config file de Pentaho 33

Índice de tablas

Tabla 6.1. Costes de Hardware 40
Tabla 6.2. Resumen de tareas y costes 41
Tabla 6.3. Resumen de costes totales 41

Capítulo 1. Introducción

1.1 ¿Por qué el Big Data?

El Big Data es una consecuencia natural de la evolución en la producción de datos que cada persona ha venido experimentado a lo largo del desarrollo de las tecnologías, desde la creación de ordenadores personales y la interconexión de estos con el desarrollo del internet. Esta generación de datos no ha parado de crecer exponencialmente desde la irrupción hace unos años de los SmartPhone hasta llegar a la hiperconectividad que tenemos hoy en día con el desarrollo de los *Wearables*, redes sociales, el 4G y la llamada tecnología disruptiva 5G que promete crear una nueva etapa en el desarrollo del internet de las cosas (IoT).

Como consecuencia directa de todas estas nuevas herramientas para generar datos la forma de entenderlos ha cambiado sustancialmente. Ya no solo es nuestro nombre, DNI o dirección de correo electrónico; podemos generar nuevos datos con fotos, enlaces, tuits, o metadatos de nuestros dispositivos electrónicos. Esto ha dado lugar a que los mecanismos tradicionales, tanto de almacenamiento como de análisis, no pudieran arrojar soluciones viables. Debido a ello, se ha desarrollado un nuevo conjunto de herramientas que, trabajando juntas, son capaces de desarrollar soluciones a esta divergencia entre el ritmo de creación de datos y la capacidad para analizarlas, dando como nacimiento a lo que se conoce actualmente como Big Data.

Por otro lado, la cantidad de personas que emigraron a la Unión Europea (UE) en 2018 fue de 2.4 millones de inmigrantes, los cuales entraron procedentes de países no pertenecientes a la EU-27. Otro dato interesante es que el 1 de enero de 2019, 21.8 millones de personas (4,9 %) de los 446.8 millones de personas que vivían en la UE no eran ciudadanos de la EU-27. Solo en 2019 el FAMI (Fondo de Asilo Migración e Integración) español gastó 437 millones de euros. Siendo estos recursos tan limitados es

esencial entender los flujos para realizar la mejor inversión posible y, teniendo la cantidad de datos masivos disponibles hoy en día y aplicando técnicas de Big Data, es factible realizar un análisis completo.

1.2 Objetivo General

Contribuir a entender los flujos migratorios de la UE mediante el análisis de los *datasets* elegidos, para que se pueda hacer un gasto más eficiente de los pocos recursos que se destinan año a año. De esta forma, se puede mejorar la integración de los migrantes en los países de destino y pueden incluirse en un menor tiempo a la economía productiva nacional beneficiando a los migrantes, los países de destino y las economías.

Cabe añadir que el desarrollo de este sistema y los resultados obtenidos al realizar los análisis de los datos no busca obtener la verdad absoluta de como invertir los recursos de inmigracion, sino dar una información veraz y científica para ayudar a las autoridades competentes a invertir de forma eficiente los recursos monetarios como de capital humano disponibles.

1.3 Objetivos Específicos

1) Creación de un ecosistema Big Data que combine los tres tipos herramientas básicas: herramienta de almacenamiento y gestión de datos, herramienta de análisis de datos y herramienta de visualización de datos; siendo capaz de recibir una entrada (consulta) y proporcionando una salida (informe).

2) Que el ecosistema sea capaz de ejecutar diversos algoritmos sobre el mismo conjunto de datos para poder realizar comparaciones entre ellos.

3) Analizar los *datasets* disponibles con datos verificados de instituciones públicas o fundaciones con las técnicas algorítmicas que mejor se ajusten a los datos, y proporcionar una salida con los resultados obtenidos.

1.4 Requisitos

En este apartado se enumeran una serie de requisitos para que el entorno a desarrollar sea útil:

1) Se requerirá que el ecosistema sea capaz de almacenar los datos a analizar y que estos estén accesibles de forma flexible para su posterior análisis.

2) Se requerirá que el ecosistema sea capaz de analizar los datos extraídos del sistema gestor de datos.

3) Se requerirá que el ecosistema sea flexible y capaz de ejecutar diversos algoritmos sobre el mismo conjunto de datos para su posterior comparación.

4) Se requerirá que el ecosistema sea capaz de realizar un informe con los resultados del análisis.

5) Se requerirá la opción de que el ecosistema sea flexible a la hora de escoger y desarrollar nuevos tipos de informes dependiendo del algoritmo seleccionado.

6) Se requerirá que el ecosistema sea capaz de manejar grandes volúmenes de datos.

1.5 Aplicación real del campo de estudio.

Del campo de estudio escogido los flujos migratorios de la unión europea podemos obtener diversas aplicaciones prácticas en la actualidad, ya que, estudiando el contexto que ocurrió hace un lustro en la UE (Guerra de Siria), la actual crisis sanitaria debido al virus SARS-COV-2 y los miles de refugiados/inmigrantes que llegan desde los países del tercer mundo, se resalta la importancia de optimizar los cada vez más los escasos recursos que destinan los gobiernos europeos a la integración de estas personas con la cultura, el idioma, y la economía.

Una rápida adaptación con el menor uso de los recursos existentes posibles permitiría la cohesión entre estos migrantes y la población nativa de

Europa. Para lograr esta optimización de recursos, es primordial responder algunas preguntas básicas analizando los datos disponibles, como puede ser: quién viene a europa, de dónde viene y a dónde suelen ir, cuál sería su mejor destino teóricamente, y cuántos van a venir en los próximos años. Para responder a estas preguntas podemos utilizar diversas técnicas de análisis de datos, como por ejemplo, *clustering* (quién viene y cómo son), árboles de decisión (a dónde ir), y predicciones (cuántos van a venir), entre otras.

1.6 Estado del arte

Se ha realizado un estudio de publicaciones y papers relacionados con la migración y, a continuación, se presentarán los trabajos que se consideran más relevantes.

A migration equilibrium model with uncertain data and movement costs

Este artículo [1], se considera un modelo de desigualdad variacional de la migración en el que las poblaciones se mueven de acuerdo con las funciones de utilidad correspondientes a varias ubicaciones y costos de movimiento. A diferencia de los modelos previamente estudiados basados en desigualdades variacionales, se tiene en cuenta la posibilidad de que algunos de los datos del modelo no sean deterministas, sino que se modelen a través de sus distribuciones de probabilidad.

Human migration networks and policy interventions: bringing population distributions in line with system optimization

En este trabajo [9], se presenta un nuevo modelo de red multiclase de migración humana que asume un comportamiento de optimización del sistema, es decir, el sistema siempre tenderá a su punto más óptimo. En nuestro caso, sería una distribución de los migrantes y aprovechamiento de los recursos.

Para ello demuestran cómo los gobiernos a través de subsidios pueden inferir en la distribución de la población para que esta sea óptima tanto a nivel de recursos como a nivel social.

Uso de Big Data para entender las migraciones

A principios de 2014, los investigadores académicos del *Queen College*

de Nueva York, el *Qatar Computing Research Institute* de Doha y la Stanford University utilizaron específicamente tuits geolocalizados. De los 500 millones iniciales, se fueron reduciendo hasta que se obtuvieron al menos 3 millones de tuits geolocalizados por país, con los que se estudiaron los tuits publicados entre un país A y un país B, donde dichos países tenían grandes flujos migratorios, y se consiguió estudiar las tendencias adaptativas en el tiempo [13].

El equipo de Facebook Data Science

Se realizó un estudio de datos agregados y anónimos de todos los usuarios de Facebook que enumeran tanto su ciudad natal como su ciudad actual en su perfil para analizar la migración coordinada[1]. El estudio destaca cómo se pueden usar los datos de Facebook para analizar la movilidad humana, en particular, con la posibilidad de mapear las migraciones internas e internacionales entre sí [13].

E-GEOS

Es una empresa que realizó un estudio de rutas migratorias con Big Data, específicamente en las rutas de África y Oriente Medio. Para ello, se utilizaron todo tipo de datos: redes sociales, datos de móviles, controles de fronteras, monitorización de conflictos, y, como valor añadido, fotos satelitales.

Todo con el cometido de optimizar los recursos de las agencias gubernamentales y no gubernamentales en beneficio de todos los migrantes, para asegurar sus derechos humanos y la ayuda humanitaria prestada. Actualmente, las opiniones de los usuarios finales alentaron a E-GEOS a trabajar en un proyecto de demostración de seguimiento, para desarrollar y demostrar los servicios propuestos en un escenario (casi) operativo [12].

[1] Un flujo de población de la ciudad A (ciudad natal) a otra ciudad B (la ciudad actual) se considera una **migración coordinada** si, entre el número total de personas que tienen como ciudad natal la ciudad A, es menor al número de personas que tienen como ciudad actual la ciudad B.

Capítulo 2. Tecnologías usadas

2.1 ¿Qué es el Big Data?

El Big Data, como su nombre indica, trata de procesar una gran cantidad de datos a una velocidad alta, y que además contengan un gran valor añadido, para poder sacar provecho de su análisis. Además, los datos a analizar tienen que ser veraces y pueden venir de una gran variedad de fuentes y formatos diferentes. Esto provoca una serie de características conocidas como las 5 V's del Big data :

- **Volumen**: se refiere a la cantidad física de información que se va a procesar generalmente expresada en magnitudes de bytes.
- **Velocidad**: hace referencia a la velocidad a la cual procesamos los datos, lo más rápido posible sin perder la veracidad de nuestros análisis (no podemos esperar 3 años en terminar un análisis ni hacerlo muy rápido sin validez científica).
- **Valor**: hace referencia a que datos a analizar tienen que aportar valor añadido. Se analizan datos útiles, como las cuentas de una empresa, segmentación de personas, etc.
- **Veracidad**: hace referencia a que los datos a analizar tienen que ser reales y fácilmente verificables (no son válidos datos creados al azar).
- **Variedad**: hace referencia a las fuentes de nuestros datos, que pueden ser desde redes sociales, hasta bases de datos convencionales, o videos y fotos.

En el Big Data se usan principalmente tres tipos de herramientas: las herramientas de almacenamiento de datos, las de análisis (donde se aplican algoritmos y técnicas), y las de visualización de datos.

2.2 Tipos de herramientas utilizadas

Herramientas de gestión de datos

Un sistema gestor de base de datos es un conjunto de programas que permiten el almacenamiento, modificación y extracción de la información en una base de datos. Los usuarios pueden acceder a la información usando herramientas específicas de consulta y de generación de informes, o bien mediante aplicaciones al efecto.

Los sistemas gestores de base de datos se dividen en dos grandes grupos llamados SQL y NOSQL.

Sistemas Relacionales (SQL)

Son sistemas basados en el **modelo relacional**, para el modelado y la gestión de bases de datos, es sobre la lógica de predicados y en la teoría de conjuntos. Tras ser postuladas sus fundamentos en 1970 por Edgar Frank Codd, de los laboratorios IBM en San José (California), no tardó en consolidarse como un nuevo paradigma en los modelos de base de datos.

Su idea fundamental es el uso de **relaciones**. Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados tuplas. Pese a que esta es la teoría de las bases de datos relacionales creadas por Codd, la mayoría de las veces se conceptualiza de una manera más fácil de imaginar, pensando en cada relación como si fuese una tabla que está compuesta por registros (cada fila de la tabla sería un registro o "tupla") y columnas (también llamadas "campos"). El lenguaje práctico que usa para implementar el modelo relacional es el **SQL** (por sus siglas en inglés *Structured Query Language*; en español lenguaje de consulta estructurada) es un lenguaje de dominio específico utilizado en programación, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales.

Sistemas No Relacionales (NoSQL)

Las bases de datos NoSQL son sistemas de almacenamiento de información que no cumplen con el esquema entidad–relación. Tampoco utilizan una estructura de datos en forma de tabla, donde se van almacenando los datos, sino que para el almacenamiento hacen uso de otros formatos como clave–valor, mapeo de columnas, basado en documentos o grafos.

Herramientas y técnicas de Análisis

Una herramienta de análisis es un programa o aplicación cuyo objetivo final es aplicar una serie de técnicas y algoritmos para poder analizar datos, ya estén almacenados en un sistema gestor de BBDD, texto plano, metadatos, imágenes, videos etc. En definitiva cualquier formato de información que sea capaz de ser analizado.

Técnicas de análisis

- **Clustering**: es una técnica de análisis de datos que consiste en agrupar los datos en conjuntos de características similares. Por ejemplo, pais de procedencia, nivel de estudios, edad, sexo, etc.

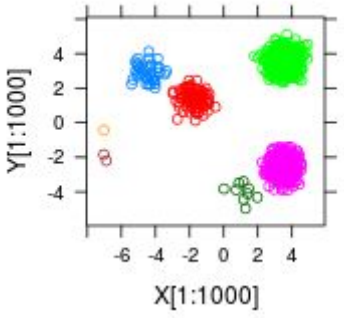


Figura 2.1. Ejemplo de clustering

- **Árbol de decisión**: es una técnica algorítmica que consiste en generar un árbol que mediante una entrada dará una respuesta binaria (sí o no), tanto la estructura del árbol como el tipo de datos dependerá del problema que se quiera resolver. Por ejemplo, se jugará un partido de tenis (sí o no).

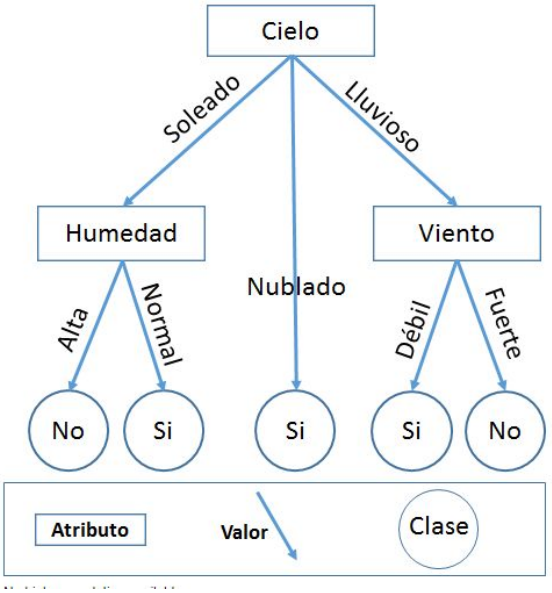


Figura 2.2. Ejemplo de árbol de decisión

- ***Machine Learning***: como su nombre indica es una técnica que consiste en enseñar a las máquinas a realizar tareas o resolver problemas, principalmente partiendo de un conjunto de datos base para entrenar a nuestro algoritmo, y que después pueda identificar casos reales, por ejemplo el análisis de sentimientos en redes sociales (Twitter).

- **Predicciones**: son técnicas estadísticas con las cuales conseguir una predicción o estimación del futuro.

- ***Deep Learning***: es un conjunto de técnicas/algoritmos que consiste, a diferencia del *machine learning*, en un aprendizaje dinámico de la inteligencia artificial autónoma por parte del programa. El mejor ejemplo que tenemos hoy en día son las redes neuronales.

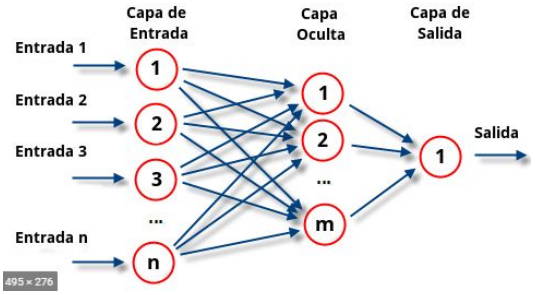


Figura 2.3. Ejemplo de *Deep Learning*

Herramientas de visualización de datos

Son herramientas especializadas en la visualización de los datos de diversas formas más amigables y comprensibles para el ser humano, facilitando el entendimiento de los resultados obtenidos al aplicar diversas técnicas de análisis a través de los que conocemos como reportes o *dashboards*.

2.3 Técnicas de desarrollo utilizadas

¿Qué es el Test Driven Development (TDD)?

Es una metodología de diseño/desarrollo de software que consiste en cambiar el orden de diseño de software tradicional, es decir, primero se escribe el código y luego se testea. Con TDD primero escribiremos los tests, los cuales fallarán al no estar implementada la funcionalidad para pasarlos, y posteriormente escribir el código que superará el test. Esto se hace con el fin de obtener un código más robusto y más libre de errores en post producción, al tener código que ha pasado un filtro de posibles errores.

¿Por qué usamos Test Driven Development (TDD)?

Usamos TDD porque es un modelo de desarrollo de aplicaciones que mejora ampliamente el tiempo de desarrollo reduciéndolo y solventando diferentes errores a la hora de programar una funcionalidad, pensando las condiciones que tiene que cumplir antes de empezar a programarla.

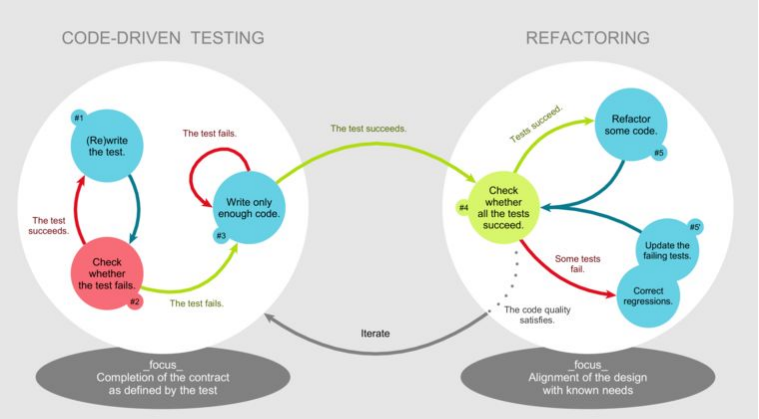


Figura 2.4. Ejemplo de TDD

¿Qué es el *Behavior Driven Development* (BDD)?

El BDD es una metodología de diseño/desarrollo de software que, al igual que el TDD, consiste en escribir primero los tests que fallarán por no tener la funcionalidad implementada, con la diferencia que el test que se desarrolla para fallar es un test que prueba el comportamiento en conjunto de dos o más elementos independientes de una aplicación. Una vez los elementos individuales han pasado por el TDD, se les prueba con BDD para evitar fallos de comunicación o comportamiento entre estos elementos en post-producción.

Un ejemplo claro sería el desarrollo de los entornos *backend* y frontend. Sus tests TDD se hacen por separado de forma independiente pero la aplicación está pensada para que trabajen en conjunto por lo que haremos un desarrollo BDD para confirmar que los elementos desarrollados individualmente son capaces de trabajar juntos y tener el comportamiento esperado.

¿Por qué usamos *Behavior Driven Development* (BDD) ?

Usamos BDD porque es un modelo de desarrollo que nos permite definir una arquitectura clara con comportamiento entre las diferentes clases. De esta forma bien estructurada y definida, tomando en cuenta las muy diferentes apps que se van a integrar en el desarrollo del TFG que se presenta, ha sido la decisión más adecuada.

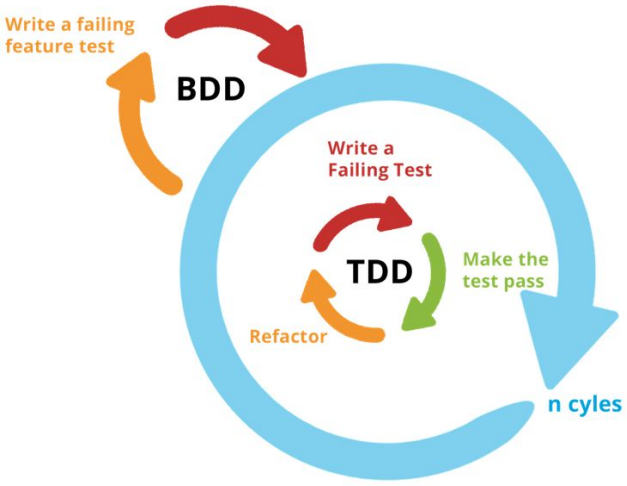


Figura 2.5. Ejemplo BDD

2.4 *Design Patterns*

¿Qué son los *Desing Patterns* o Patrones de Diseño?

Es una forma específica de escribir código, definir comportamientos o definir estructuras que nos ayudan a dar una solución validada a un problema, simplificando la interacción entre los diferentes componentes que forman parte de nuestra solución, y creando soluciones más robustas y fáciles de leer.

Patrones usados en este TFG

- **Patron de diseño *Singleton***: consiste en crear una única instancia de un objeto determinado en nuestra solución, y que esta sea accesible o modificable desde otros objetos de la solución.

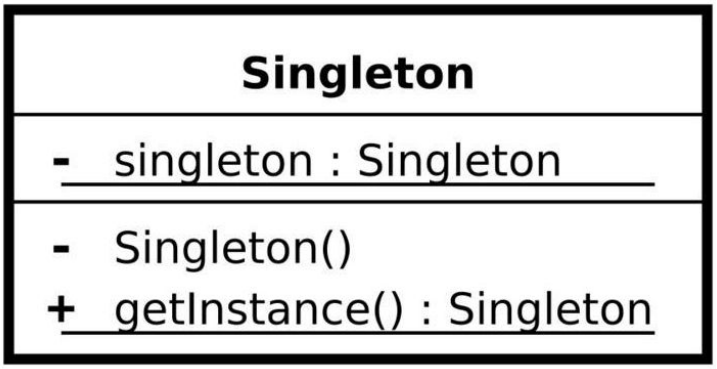


Figura 2.6. Ejemplo de *Singleton pattern*

- **Patrón de diseño *Adapter***: permite a dos clases con diferentes interfaces trabajar entre ellas, a través de un objeto intermedio con el que se comunican e interactúan.

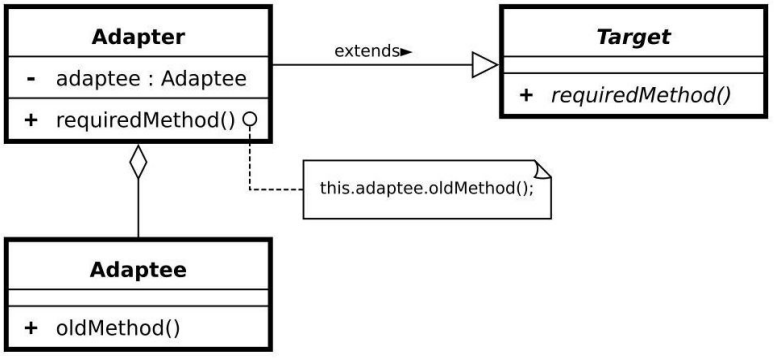


Figura 2.7. Ejemplo de *Adapter pattern*

- **Patrón de diseño MVC**: es un patrón que consiste en separar la responsabilidad de gestionar nuestra solución en tres grupos muy concretos.
    - **Modelo**: es el encargado de gestionar los datos, su mantenimiento, extracción, etc.
    - **Vista**: es el encargado de gestionar cómo se muestra la información y la lógica detrás de esta.
    - **Controlador**: es el encargado de interactuar con el usuario y enviar las diferentes peticiones al modelo o a la vista.

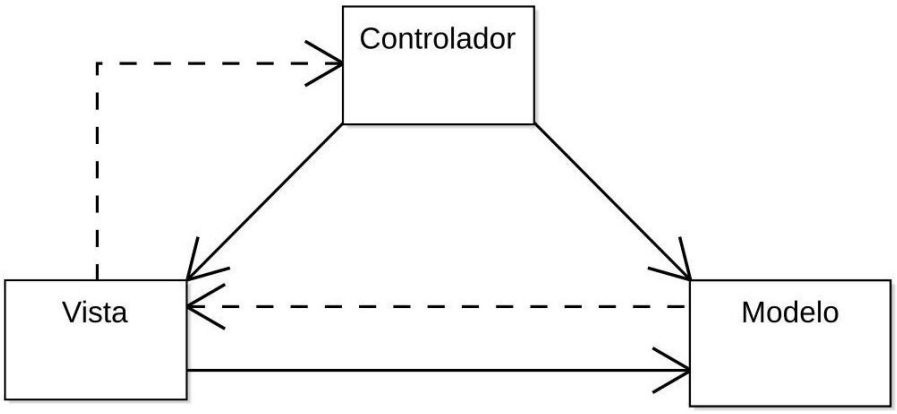


Figura 2.8. Ejemplo de *MCV architecture*

¿Por qué usamos los Patrones de Diseño?

Los usamos debido a que está ampliamente demostrado que estos dan soluciones más ágiles y resistentes a fallos cuando se usan en el desarrollo de aplicaciones. Esto se debe principalmente a los muchos años que se llevan desarrollando aplicaciones, en las que han surgido diferentes problemas a diferentes personas, y se han hallado diferentes soluciones "óptimas" que cumplen con las necesidades de la aplicación.

2.5 Java

¿Qué es Java?

Es un lenguaje de programación orientado a objetos multiplataforma, interpretado y compilado. Es uno de los más usados a lo largo del mundo, dado que es posible ejecutar una misma aplicación en distintos sistemas operativos con una misma base de código, debido que el mismo se ejecuta en una máquina virtual.

¿Por qué usamos Java ?

Usamos Java para este TFG debido a que el código fuente de Hadoop está escrito en Java dando bastante facilidades para implementar algoritmos en este lenguajes. Además, las otras herramientas, MongoDB y Pentaho tienen librerías potentes escritas en Java. En el caso de MongoDB, la MongoDB Foundation proporciona un curso gratuito de cómo utilizar la API, desarrollada por ellos en este lenguaje.

2.6 Software utilizado

Maven

Es un software [7] que permite administrar de forma rápida, simple y eficiente todos los procesos de creación de una aplicación desde las etapas primarias de validación, tests y *build*. Además, permite agregar de forma simple dependencias, o *plugins* externos a nuestro proyecto de manera que son completamente utilizables una vez los hayamos añadido en el fichero POM (*Project Object Model*) en formato XML (*eXtensible Markup Language*, Lenguaje de Marcas Extensible)

Este fichero `pom.xml` es la representación de un proyecto Maven escrito en formato XML. Con él podemos importar plugins, configurar propiedades de compilación de nuestro proyecto, como la versión del compilador a utilizar, si esta será codificada en UTF-8 e incluso dependencias externas simplemente haciendo referencia a estas con las etiquetas adecuadas.

- `Repositories`: indica las propiedades de los repositorios de dependencias a utilizar.
- `Repository`: dentro se colocan las propiedades de los repositorios
- `Id` : Id del repositorio a utilizar.
- `Name`: nombre del repositorio.
- `URL`: indica el repositorio a utilizar.
- `Dependencies`: indica las propiedades de las dependencias a utilizar.
- `Dependency`: contiene las propiedades de las dependencias externas a utilizar.
- `Groupid`: indica el proyecto al que pertenece un *artifact*.
- `Artifactid`: indica un conglomerado de librerías únicas entre proyectos del mismo `groupid`.
- `Version`: indica la versión del *artifact* a utilizar.

```xml
<project>
  ...
  <properties>
    <mavenVersion>3.0</mavenVersion>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-artifact</artifactId>
      <version>${mavenVersion}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-core</artifactId>
      <version>${mavenVersion}</version>
    </dependency>
  </dependencies>
  ...
</project>
```

Figura 2.9. Ejemplo de `pom.xml`

¿Por qué utilizamos Maven?

Se utilizó Maven debido a la variedad y cantidad de librerías externas a utilizar para este TFG. Debido a ello, la complejidad para hacer o encontrar las librerías correctas a utilizar resultaría en un exceso de dificultad. Aparte de resolver este problema, Maven es capaz de lanzar tests unitarios con un simple comando, y está perfectamente integrado en el IDE utilizado para desarrollar este TFG (Eclipse). Todos estos puntos fuertes nos hicieron decantarnos por Maven como herramienta de administración del ciclo de vida del proyecto.

MongoDB

MongoDB [8] es un software gestor de bases de datos no relacional, basado en colecciones (mecánica parecida a una tabla en sistemas SQL) y documentos (mecánica parecida a los registros en sistemas SQL) que son almacenados en formato BSON (escritura en binario de un objeto en formato JSON, *JavaScript Object Notation*).

Figura 2.10. MongoDB

JSON es un formato sencillo para el intercambio de información que consiste en una definición de un par clave:valor separados por puntos y puestos entre llaves.

```
{
        "_id" : ObjectId("573f7197f29313caab89b222"),
        "sku" : 20000062,
        "name" : "New Year's Day [Single] - CD",
        "type" : "Music",
        "regularPrice" : 20.99,
        "salePrice" : 20.99,
        "shippingWeight" : "0.25"
}
```

Figura 2.11. Ejemplo de fichero JSON

¿Por qué utilizamos MongoDB?

Se tomó la decisión de utilizarlo debido a la utilidad de poder beber de diferentes fuentes de datos con esquemas variados en un mismo sistema de almacenamiento de información, además de ser software libre y la amplia documentación y cursos oficiales totalmente gratuitos que hacen posible usar el sistema gestor de forma cómoda.

¿Qué es el software de agregación?

Es un *framework* que permite realizar agregaciones en los datos tales como agrupaciones, medias, sumas, etc. Este *framework* se divide en *stages* parecido a los *pipes* de la línea de comando bash en Linux, donde la salida de uno es la entrada del siguiente comando. Esto permite pasar el resultado de un *stage* a otro para ir realizando operaciones sobre este.

```
db.movieDetails.aggregate([{$match : { imb :"tt0758652"  },{$unwind : "$tomato"}}])
```

Figura 2.12. Ejemplo de *query* en *aggregation framework*

¿Por qué utilizamos el *aggregation framework*?

Utilizamos el *aggregation framework* de MongoDB debido a que puede ser interesante dar distintas entradas a nuestro algoritmo de análisis para comprobar los resultados de este.

Hadoop

Es un *framework* de código libre [4] que permite el análisis en paralelo de datos en sistemas distribuidos con hardware comercial (bajo en prestaciones) usando el sistema de ficheros distribuido HDFS [5] y el algoritmo MapReduce, y cuenta con un amplio ecosistema de herramientas que lo complementan .

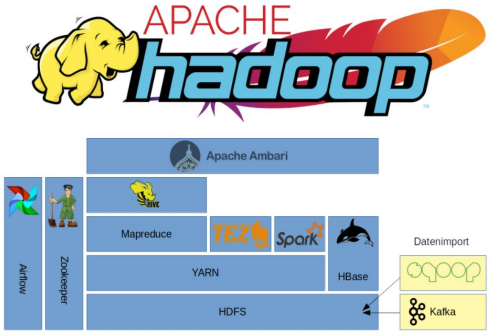


Figura 2.13. Hadoop Ecosystem

- ¿Qué es HDFS (Hadoop Distributed File System)?

  Es un sistema de ficheros distribuidos ofreciendo sistemas escalables resistentes a fallos debido a su característica de replicación de datos. Para ello, HDFS se basa en un sistema maestro esclavo donde el maestro (el NameNode) es el único encargado de gestionar los ficheros y los metadatos (estos contienen información sobre los ficheros y donde se encuentran entre los esclavos) y el esclavo (DataNode) son los encargados de almacenar los ficheros que por defecto tienen un tamaño de 64 MB.

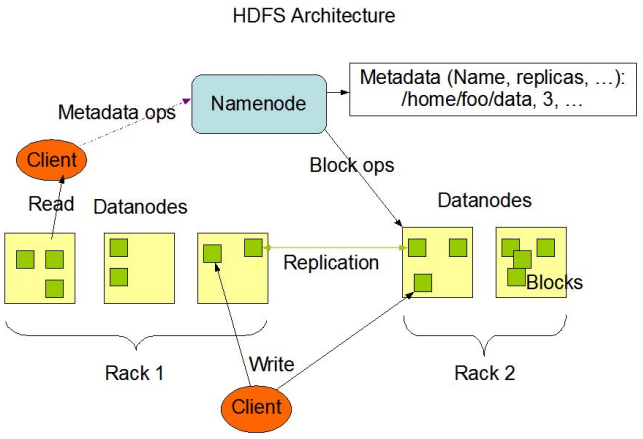


Figura 2.14. Ejemplo de *HDFS architecture*

- ¿Por qué usamos HDFS?

Usamos HDFS porque es el formato de ficheros distribuidos con el que trabaja Hadoop por defecto.

- ¿Qué es MapReduce?

MapReduce es un framework diseñado para trabajar con grandes cantidades de datos en formato distribuido y procesamiento paralelo que sigue una arquitectura de master-slave. Toma su nombre de las dos principales funciones que realiza el Map y Reduce se desarrollará de forma más específica estos términos en los apartados posteriores.

- Nodo Master o JobTracker

Consiste en un nodo o JobTracker que es el encargado de manejar todo el *workflow* de un *job* en el cluster de Hadoop, siendo capaz de recibir peticiones de los clientes, programar tareas (TaskTracker), indicar el estado de los *jobs* actuales y procesar los metadatos de los ficheros.

- Nodo Slave o TaskTracker

Consiste en uno o más nodos encargados de almacenar toda la información a analizar, ejecutar las solicitudes de trabajo del jobTracker, ejecutar el código proporcionado para analizar la información almacenada, realizar la configuración adecuada para llevar a cabo el trabajo solicitado y es el encargado de suministrar la salida del trabajo al JobTracker, así como su actualizacion o modificacion.

- Función Map

La función Map se encarga del mapeo de los datos, es decir recibe una entrada estándar y da como salida listas de pares

clave:valor, y después se agrupan todos los valores coincidentes en el campo clave.

```
Map (clave1, valor1) -> lista (clave2, valor2)
```

- Función Reduce

  La función Reduce consiste en procesar paralelamente las agrupaciones que nos da la función Map de salida. Esto significa que se recibirá una única clave con una lista de valores asociados a ella para poder realizar operaciones de agregación, y así obtener un conjunto más pequeño de datos dando estos como salida.

```
Reduce (clave2, lista(valor2)) -> lista(valor2)
```

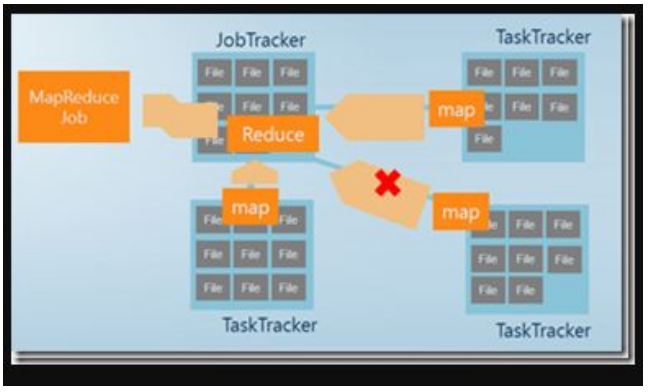


Figura 2.15. Ejemplo de MapReduce

¿Por qué usamos Hadoop?

Decidimos usar Hadoop debido a su compatibilidad natural con el lenguaje de programación en el que está desarrollada la infraestructura Java, la gran variedad de APIs escritas en Java para usar Hadoop y sus elementos como el HDFS o YARN. Además de la documentación extensa en internet sobre las APIs con varios ejemplos de uso y porque es la herramienta almacenamiento y de análisis masivo de datos que mejor se adapta al tipo de datos que vamos a analizar canalizado en lotes más que en tiempo real.

¿Qué es YARN (*Yet Another Resource Negotiator*)?

Es el framework [12] que permite a Hadoop poder ejecutar varios motores de ejecución, entre ellos el MapReduce. Se creó principalmente

para ampliar la funcionalidad de Hadoop ya que funcionaba de forma irregular al ejecutar algoritmos que no estuvieran enfocados en el MapReduce. YARN consta de tres elementos principales: *Resource Manager*, *Node Manager* y el *Application Master*.

- Resource Manager

  Es el encargado de gestionar los recursos de un cluster Hadoop. Tiene dos componentes: el *scheduler*, encargado de asignar los recursos de hardware a los *jobs*. El scheduler no es capaz de monitorizar el estado del *job* ni de asegurar su correcta ejecución. El otro componente clave es el Application Manager, el cual es el encargado de aceptar las peticiones de trabajos proporcionadas por los clientes, además de negociar el contenedor en el que se ejecutará la aplicación, siendo capaz de reiniciar los trabajos si fuera necesario debido a errores.

- Node Manager

  Es el encargado de gestionar los *jobs* con información proporcionada por el Resource Manager y también asigna los recursos disponibles a los *jobs* en forma de contenedores (conjunto de recursos con CPU, RAM, etc) para que las aplicaciones se ejecuten.

- Application Master

  Es el responsable de negociar los recursos apropiados con el Resource Manager y monitorizar su estado y su progreso. También coordina la ejecución de todas las tareas en las que puede dividirse su aplicación.

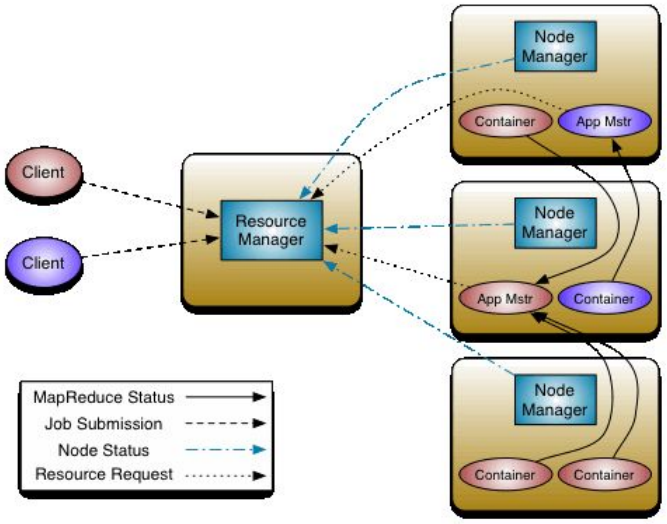


Figura 2.16. Ejemplo de arquitectura en YARN

¿Por qué usamos YARN?

Decidimos usar YARN para abstraernos totalmente de la problemática que surge al pensar cómo repartir eficientemente los recursos entre los diferentes *jobs* o aplicaciones que vayamos a ejecutar, siendo este el *framework* recomendado para ejecutar aplicaciones con Hadoop. Viniendo por defecto con la instalación del software de Hadoop, nos pareció conveniente aprovechar las herramientas a nuestra disposición y no "reinventar la rueda".

Pentaho

Pentaho [10, 11] es una herramienta de *Business Intelligence* que concentra una gran variedad de herramientas que van desde los ETL, hasta la creación de dashboards y reportes.

Figura 2.17. Logo de Pentaho

¿Por qué usamos Pentaho?

Decidimos usar Pentaho por la API de creación de informes desarrollada para Java (`pentaho-reporting-engine`) que permite crear reportes programables con facilidad.

2.7 K-means

Es un algoritmo de clustering no supervisado que consiste en ir agrupando los puntos en conjuntos llamados clusters de una nube de puntos que tomamos como entrada al algoritmo. Para realizar esta analitica el k-means sigue los siguientes pasos.

1) K-means necesita que le proporcionamos el número de clusters a crear en nuestro dataset. Para saber exactamente cuál es el número de k-clusters óptimo podemos usar el método del "codo". Se representa en una gráfica la suma de los cuadrados de la variación intra-cluster y se

suele buscar el número de cluster que no baja significativamente el nivel de variación.

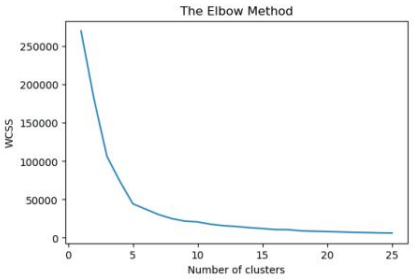


Figura 2.18. Método del codo

2) Una vez tenemos la cantidad de grupos a utilizar, se calcula aleatoriamente tantos centroides (medias) como grupos haya.

3) Se clasifican los datos calculando la distancia entre los centroides y cada punto de la nube, agrupando los que más cerca se encuentran de los centroides escogidos.

4) Se repite el proceso para un número N de iteraciones o cuando los conjuntos no cambien demasiado después de un número N-m iteraciones, siendo m el número de iteraciones ya ejecutadas.

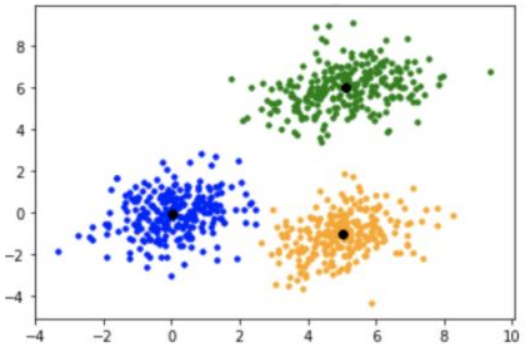


Figura 2.19. Ejemplo de k-means

Capítulo 3. Desarrollo del ecosistema y flujo de trabajo

En este capítulo se explicará la composición del ecosistema Big Data desarrollado y a continuación se explicara el flujo de trabajo del ecosistema.

3.1 Clases desarrolladas

A continuación se nombraran y explicarán las clases desarrolladas y su función para el ecosistema Big Data.

- **BigDataEnviroment:** es la clase principal de nuestro ecosistema, lleva nuestra función `main` y, por lo tanto, es la encargada de recibir los parámetros desde la línea de comandos y pasarlos a nuestra clase que implementa el patrón de diseño MCV.

- **ModelViewController**: como su nombre indica, es la clase encargada de implementar el patrón de diseño MCV, siendo al mismo tiempo el controlador de este ecosistema. Contiene como atributos las clases LogicEnviroment y GraphicsEnviroment. Además extiende la clase JFRAME que será la ventana principal de nuestra interfae gráfica e implementará las funciones de ActionListener para controlar las interacciones del usuario con el entorno.

- **LogicEnviroment**: es la clase que contiene los objetos donde se implementa el *backend* de nuestro ecosistema, encargada de parsear los argumentos que llegan desde la línea de comandos dirigidos al *backend* y pasarlos a sus respectivos destinos. Contiene como argumentos a las clases DatabaseAnalizerLinkClass y AnalizerReportLinkClass encargadas de implementar el patrón *Adapter*.

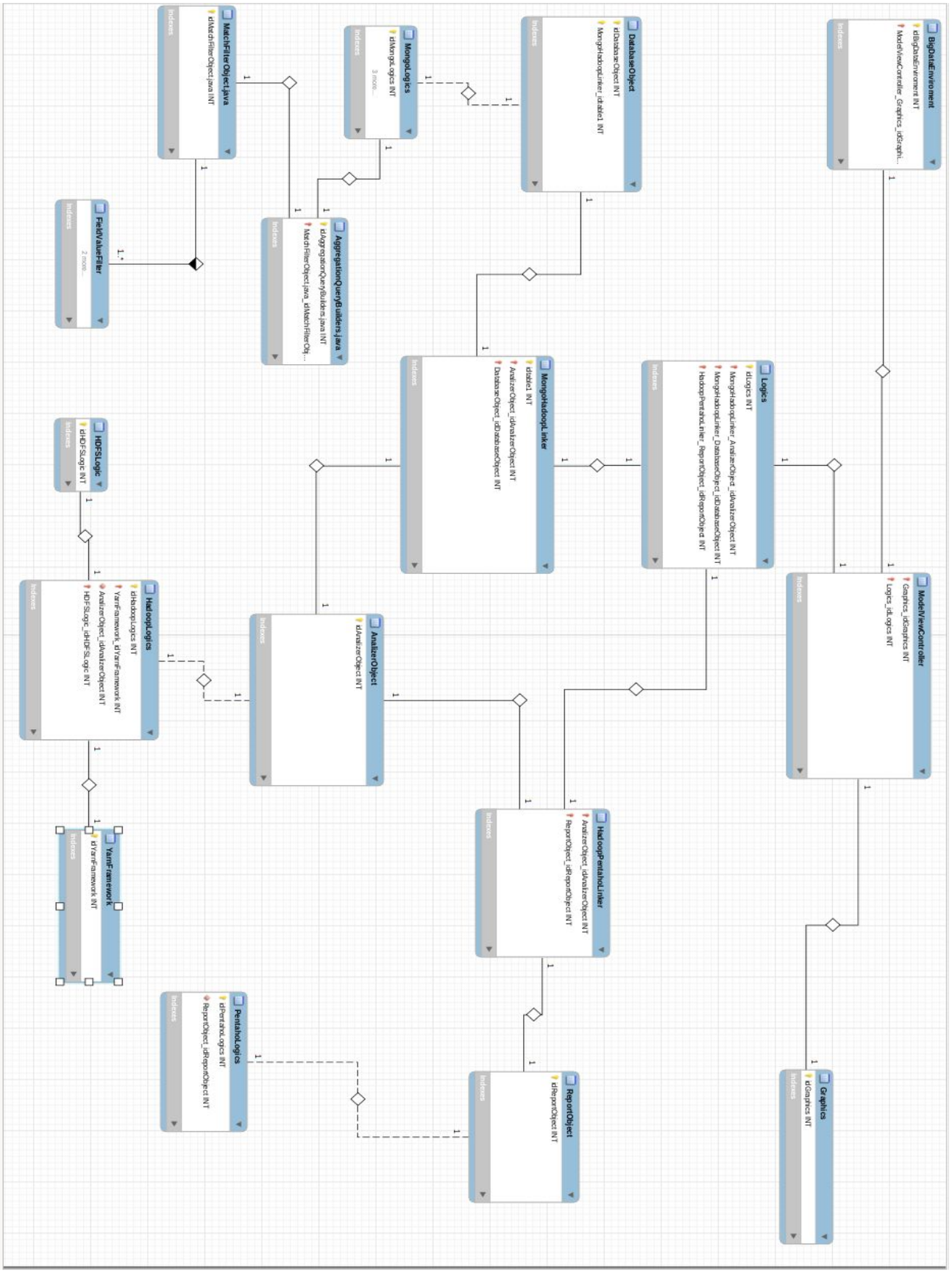


Figura 3.1. Ecosistema Big Data en UML

- **DatabaseAnalizerLinkClass**: es la clase encargada de hacer de enlace entre nuestro sistema gestor de bases de datos y nuestro analizador, en esta clase implementamos el patrón *Adapter* con la ayuda de las clases DatabaseObject y AnalizerObject.

- **DatabaseObject**: es una clase abstracta que nos permite homogeneizar mediante la herencia los sistemas gestores de bases de datos (en nuestro proyecto MongoDB). Así logramos que mediante la implementación de esta clase nuestro ecosistema pueda cambiar libremente de sistema gestor de bases de datos sin que esto afecte a la ejecución del entorno.

- **MongoLogic**: es una clase que hereda de DatabaseObject, y es la encargada en nuestro ecosistema de conectar con la base de datos, realizar las extracciones de datos mediante la clase AggregationQueryBuilder y de exportar los datos obtenidos al fichero para analizar.

- **AggregationQueryBuilder**: es la clase encargada de implementar el creador de querys utilizando las *pipelines* y operadores del *aggregation framework* de MongoDB para ello cuenta con funciones y clases encargadas de crear los diferentes *pipelines* como por ejemplo la clase MatchFilterObject.

- **MatchFilterObject**: es la encargada de crear el *stage* Match que es equivalente a la cláusula `where` en el lenguaje SQL. Para ello necesita recibir tres argumentos básicos por consulta.

    a. **Filter**: se refiere al filtro que se va a utilizar para realizar la *query*. Por ejemplo, el operador `$eq` en MongoDB significa “igual que”.
    b. **Field**: es el nombre del campo al que se hace referencia en la *query*. Por ejemplo, queremos encontrar a todas las personas con el nombre de “Maria”, donde nombre es nuestro *field*.
    c. **Value**: es el valor del campo al que le queremos aplicar el *filter* escogido en la *query*. Por ejemplo, podemos usar el operador `$gte`, que significa “mayor o igual que”, combinado con un campo edad, y queremos que el valor sea 18. Con esto nos daría el resultado de las con edades mayores o iguales a 18 años.

  Para restarle complejidad al algoritmo que realiza la construcción de estas consultas y con vistas a que nuestro ecosistema soporte una interfaz gráfica interactiva se ha creado la clase FieldValueFilter.

- **FieldValueFilter**: es la clase encargada de almacenar en memoria las *querys* que se han construido hasta el momento.

- **AnalizerObject**: es una clase abstracta que nos permite homogeneizar mediante la herencia los sistemas de análisis de datos (en nuestro proyecto Hadoop). Así logramos que mediante la implementación de esta clase nuestro ecosistema pueda cambiar libremente de sistema analizador de datos sin que esto afecte a la ejecución del entorno.

- **HadoopLogic**: es una clase que hereda de AnalizerObject, y es la encargada en nuestro ecosistema de ejecutar los algoritmos y realizar la gestión del entorno distribuido de los datos. Para ello, dispone de las clases HDFSLogic y YARN.

- **HDFSLogic**: es la clase encargada de administrar el sistema de ficheros distribuido. Para ello, cuenta con la implementación de diversas funciones con las operaciones básicas de administración de sistemas, como dar permisos, crear ficheros, realizar cargas y descargas de datos.

- **YARN**: es la clase encargada de invocar al programa YARN y ejecutar los algoritmos con los que se desea analizar los datos.

- **AnalizerReportLinkClass**: es la clase encargada de hacer de enlace entre nuestro sistema de análisis de datos y nuestro sistema de reportes. En ella implementamos el patrón *Adapter* con la ayuda de la clase ya mencionada AnalizerObject y la clase ReportObject.

- **ReportObject**: es una clase abstracta que nos permite homogeneizar mediante herencia los sistemas de *reporting* (en nuestro proyecto Pentaho), y así logramos que nuestro ecosistema pueda cambiar libremente de sistema de reporting sin que esto afecte a la ejecución del entorno.

- **PentahoLogic**: es la clase encargada de implementar el sistema de *reporting* de nuestro ecosistema. Consiste en una clase que crea *reportings* programables y personalizados para representar la salida de nuestro sistema analizador de datos de una forma más amigable y comprensible para las personas.

- **GraphicsEnviroment**: clase encargada de gestionar el apartado gráfico (*frontend*) de nuestro ecosistema Big Data.

3.2 Flujo de trabajo Ecosistema Big Data.

En este apartado se describe el flujo de trabajo del ecosistema y cómo las clases que lo componen interactúan entre sí.

- **BigDataEnviroment**: es la clase donde empieza nuestro flujo de trabajo. Recibe de línea de comandos los siguientes argumentos:

    a. **-u**: se refiere al nombre de usuario con el que se conectará al sistema gestor de bases de datos.

    b. **-p**: se refiere a la password del usuario que se conectará al sistema gestor de bases de datos.

    c. **--MONGO**: indica la localización del fichero de configuración para el *backend* de MongoDB.

    d. **--HADOOP**: indica la localización del fichero de configuración para el backend de hadoop.

    e. **--PENTAHO**: indica la localización del fichero de configuración para el *backend* de Pentaho.

    f. **--MONGOCONFIG**: con este argumento se solicita al entorno que enseñe las instrucciones para crear un fichero de configuración para el *backend* de MongoDB.

    g. **--HADOOPCONFIG**: con este argumento se solicita al entorno que enseñe las instrucciones para crear un fichero de configuración para el *backend* de Hadoop.

    h. **--PENTAHOCONFIG**: con este argumento se solicita al entorno que enseñe las instrucciones para crear un fichero de configuración para el *backend* de Pentaho.

Una vez comprobado que se recibe al menos 2 argumentos por la línea de comandos, se crea el objeto ModelViewController y se le pasan los argumentos a su constructor para seguir con el flujo de trabajo.

- **ModelViewController**: es el controlador de nuestro ecosistema encargado de la comunicación entre el *backend* y el *frontend* además de implementar las interacciones con el usuario. Actualmente solo tiene como función pasar los argumentos de entrada a cada una de los objetos LogicEnviroment y GraphicsEnvironment.

- **LogicEnviroment**: implementa el *parser* de los argumentos entregados desde la línea de comandos, y se lo envía a los objetos que se encargan de gestionar el *backend* del ecosistema. Además, confirma cuáles de los objetos que han sido creados y ejecuta el *workflow* del ecosistema una vez se han terminado de cargar las configuraciones.

    a. Primero, el constructor inicializa las variables necesarias para parsear los argumentos.

    b. Segundo, recorremos el vector de argumentos pasado por línea de comandos. En el caso de MongoDB, leemos usuario, contraseña, y la ruta del fichero de configuración.

    c. Se pasa el fichero de configuración básico para MongoDB y este debe seguir un orden específico para su correcta lectura.

        - Primera línea: IP del servidor al que nos vamos a conectar

        - Segunda línea: base de datos a la que nos vamos a conectar.

        - Tercera línea: *collection* sobre la cual vamos a ejecutar las *querys*.

        - Cuarta línea: ruta absoluta donde queremos dejar el fichero donde estarán los datos exportados.

        - Quinta línea: nombre del fichero resultante de la exportación de los datos.

- Sexta línea: si es un número, indica el límite máximo de documentos que se exportarán; si es un carácter indica que se desea exportar la totalidad de los documentos resultantes de la *query* expresada.

- Séptima línea: a partir de esta línea se juega con un conjunto de etiquetas que indican las querys a construir.

    1. Etiqueta **<match>**: indica el comienzo de la definición de el *stage match* del *aggregation framework*, y la siguiente línea después de esta etiqueta se indica en el siguiente orden los campos necesarios para la *queries* separados por comas, filtro, campo y valor. Ejemplo:
       Filtro,nombre del campo, valor del campo
       `eq,sexo,masculino`

    2. Etiqueta **</match>**: esta etiqueta es la encargada de cerrar el *stage match* dando por acabado.

    3. Etiqueta **<projection>**: es la encargada de dar comienzo al *stage projection* del *aggregation framework*, y en la línea siguiente a esta etiqueta se colocará el nombre del campo que se quiera proyectar y un booleano con valor true si se quiere proyectar este campo, o false si se quiere ignorar. Ejemplo:
       nombre del campo, si queremos proyectar o no.
       `edad,false`

    4. Etiqueta **</projection>**: esta etiqueta es la encargada de cerrar el *stage projection*, dándolo por acabado.

```
127.0.0.1:27017   // Ip del servidor
test              // base de datos a usar
products          // collection a usar
/home/alberto/Escritorio/ // path del fichero a exportar
outputTestParser  // nombre del fichero
100               // numero de documentos a exportar
<match>           // inico match stage
eq,type,"Music"   // filtro,campo,valor
gte,regularPrice,14.99 //filtro,campo,valor
</match>          // fin match stage
<projection>      // inicio projection stage
type,true         // campo , excluir o no
</projection>     // fin projection stage
```

Figura 3.2. Ejemplo de *config file* en MongoDB

d. Se crean los objetos para el *aggregation framework*.

e. Se construye la cadena de conexión con los datos proporcionados y nos conectamos con el cliente a la BBDD.

f. Se pasa el fichero de configuración básico para Hadoop, y este debe seguir un orden específico para su correcta lectura.

    1. La primera línea debe tener la IP del sistema HDFS con el que conectaremos.

    2. La segunda línea debe tener la ruta donde se encuentra YARN en nuestro equipo.

    3. La tercera línea debe indicarse la carpeta remota dentro del sistema HDFS donde se subirán los datos.

    4. La cuarta línea debe indicar la carpeta donde se dejará la salida del algoritmo a ejecutar.

    5. En la quinta línea ha de indicarse el JAR file a ejecutar (nuestro algoritmo).

    6. En la sexta línea ha de indicarse las opciones a recibir por el JAR file que se ejecutará.

```
hdfs://localhost:9000 // IP DEL HDFS al que conectar (master)
/home/hadoop/hadoop-2.8.5/bin/yarn // home del programa Yarn en local
/albertoHome/ // folder remoto donde se subiran los datos
/albertoHome/outputFileTest // folder de salida del algoritmo
/home/hadoop/hadoop-2.8.5/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.5.jar // jar file a usar
wordcount // opciones del jar file a usar.
```

Figura 3.3. Ejemplo *config file* de Hadoop

g. Se pasa el fichero de configuración básico para Pentaho, y este debe seguir un orden específico para su correcta lectura

    1. La primera línea indicará el nombre del informe.

    2. La segunda línea indica el delimitador para el parser de entrada.

    3. La tercera línea indica el tipo de informe que se quiere.

    4. A partir de la cuarta línea empiezan las opciones del parser. La etiqueta **<table>** indica que se quiere un informe del tipo tabla.

    5. En la línea siguiente a la etiqueta **<table>** vendrá el nombre de las variables de la tabla separadas por comas.

    6. En la siguiente línea la variable que se mostrará en el eje Y, siempre ha de ser del tipo numérica.

    7. En la siguiente línea la variable que se mostrará en el eje X.

    8. Luego vendrá la etiqueta **</table>** encargada de cerrar el apartado.

    9. Luego vendrán la etiqueta **<inputData>** la encargada de proporcionar los directorios y nombre del fichero por si no se ejecuta el analizador previamente al informe.

    10. En la siguiente línea a la etiqueta **<inputData>** vendrá el directorio de salida del fichero.

    11. En la siguiente línea vendrá el nombre del fichero.

12. Luego vendrá la etiqueta **</inputData>** encargada de cerrar el apartado.

13. Luego vendría la etiqueta **<seriesColor>** que indicara la serie de colores a usar en el formato de salida escogido y separados por comas.

14. En la siguiente línea vendrá el vector de colores en formato hexadecimal y separados por comas.

15. En la siguiente línea vendrá la etiqueta **</seriesColor>** encargada de cerrar este apartado.

```
outputReportTest // nombre del reporte
\t // delimitador del fichero de datos de entrada
Table // tipo de representacion de datos.
<Table> // etiqueta tipo de formato
Palabras,Cantidad // nombre de las vairables  de la tabla.
Cantidad // columna que se mostrara en el Eje Y Nota: siempre variable numerica en
Palabras // columna que se mostrara en el eje X
</Table> // cierre del formato
<inputData> // etiqueta para los datos de entrada opcional
/home/alberto/Escritorio/outputTEST1 // folder de entrada
part-r-0000 // nombre del fichero de datos
</inputData> // etiqueta de cierre de los datos de entrada
<SeriesColor> // etiqueta de las series de colores opcional
#DEB887,#0000FF,#00FFFF // colores en formato hexadecialm separado por comas
</SeriesColor>  // cierre de las series de colores
```

Figura 3.4. Ejemplo *config file* de Pentaho

h. Una vez se ha terminado de cargar la configuración de entrada para los distintos objetos del *backend,* se ejecuta la función executeWorkFlow() que, dependiendo de los parámetros de entrada, ejecutará diferentes funciones de los objetos linked.

i. Para realizar el trabajo entre nuestro objeto de base de datos y nuestro objeto analizador, se llama al objeto DatabaseAnalizer y este ejecuta la función executeWorkFlowDatabaseAnalizer(), encargada de ejecutar la consulta sobre la base de datos, exportarlo a un fichero, subirlo al sistema HDFS, ejecutar el algoritmo sobre los datos y descargar la solución a local.

j. Por último se invoca al objeto AnalizerReport, que ejecuta la función makeReportAnalizerWorkFlow(). Esta función es la

encargada de crear el reporte específico y previamente programado.

3.3 Tratamiento de Datos

En casi cualquier proceso para realizar un análisis de datos tendremos que realizar una ETL (*Extract, Transform and Load*). Esto se debe principalmente a que hoy en día los datos vienen en una gran variedad de formatos, siendo el formato de un dato en la fuente diferente que en destino. Un ejemplo claro es si buscamos datos de una página web, red social, etc, estarán en formato HTML y, en el mejor de las casos, en formato JSON. Si queremos extraer esos datos (*extract*) y guardarlos en una base de datos SQL deberemos cambiar el formato de estos datos (*transform*) a uno aceptado por la base de datos y almacenarlo posteriormente (*load*).

Otro de los ejemplos claros donde tenemos que realizar una ETL de los datos es al utilizar los algoritmos de análisis, ya que los algoritmos por lo general solicitan que los datos estén en un formato específico para su correcta ejecución. En el caso del algoritmo explicado anteriormente, el k-means, solicita que los datos estén en formato de matriz donde cada línea sean un vector de puntos (datos de entrada).

Como podemos entender, el k-means necesita explicitamente datos del tipo numérico para poder realizar su ejecución, pero que pasaría si tenemos variables del tipo cualitativo que describan rasgos como el sexo, color de pelo, nacionalidad, etc. Aquí tendremos que realizar una transformación de los datos del tipo cualitativo al cuantitativo. Por ejemplo, en la variable sexo podriamos hacer la transformacion de hombres es igual a 0 mujeres igual a 1. Con esto lograríamos que el k-means analizara las variables cualitativas.

Una vez realizada esta pequeña explicación sobre un proceso fundamental en el análisis de datos como los procesos ETL, y a pesar de que nos ha sido imposible contar con datos oficiales debido a las restrictivas normas europeas, en las cuales se necesita pedir permiso para acceder a los datos en bruto para el análisis de los flujos migratorios, y cumpliendo la condición de que los datos sean utilizados para realizar estudios académicos avalados por instituciones reconocidas con este propósito, se tardaría entre 8 y 10 semanas en dar el acceso a dichos datos.

Pero, si tenemos datos agregados entrando al portal de EUROSTATS [3], estos datos agregados no son útiles para realizar análisis pero sí nos permite inferir parte de ellos en bruto logrando predecir qué tipo de transformaciones en los datos debemos realizar si quisiéramos analizarlos con el algoritmo k-means.

Los datos agregados sobre inmigracion disponibles en el portal de EUROSTATS [3] vienen separados por los siguientes temas:

- *Inmigration by age and sex*
- *Inmigration by age grout sex and citizenshit*
- *Immigration by age group, sex and country of birth*
- *Immigration by age, sex and broad group of citizenship*
- *Immigration by age, sex and broad group of country of birth*
- *Immigration by sex, citizenship and broad group of country of birth*
- *Immigration by sex, country of birth and broad group of citizenship*
- *Immigration by age group, sex and country of previous residence*
- *Immigration by age group, sex and level of human development of the country of citizenship*
- *Immigration by age group, sex and level of human development of the country of birth*
- *Immigration by age group, sex and level of human development of the country of previous residence*
- *Immigration by broad group of country of previous residence*

De los ítems mostrados, en estos temas podemos inferir que necesitaremos hacer varias transformaciones de datos para las variables *sex*, *citizenship*, y *country of birth*. La mayoría de ellos son una variable calificativa que tiene como resultado el nombre de un país. Esta transformación sería relativamente sencilla creando una función encargada de recibir un país y asignarle un número previamente establecido. Cuando se acabe el análisis del k-means, tendremos que transformar los números anteriormente asignados a variables cualitativas nuevamente.

Capítulo 4. Conclusiones y líneas futuras

Los flujos migratorios siempre existirán en nuestras sociedades, y el origen de estos tienen diferentes razones, pero el denominador común es que todo el que emigra busca una vida mejor que en su lugar origen. Existen diversas barreras que pueden dificultar la inmigracion como políticas migratorias, la cultura o el idioma. Y estas barreras generan un gran sobrecoste en el presupuesto para inmigracion e integracion de la Unión Europea, por lo que saber que tipo de personas (*clusters*) vienen podría mejorar significativamente el rendimiento de las inversiones en este aspecto, ya que nos indicará de forma general algunas de las necesidades de los migrantes que pudiéramos estar pasando por alto por no hacer un análisis de los datos.

Desde un comienzo, este ecosistema se ha planteado para que sea flexible, fácil de mantener y de agregarle funcionalidades. Por eso se han utilizado diversos patrones de diseño con el fin de que los objetos finales de este entorno sean más como *plugins* que se puedan poner y quitar a conveniencia de la persona que vaya a utilizar el ecosistema, dando total libertad como flexibilidad. Por ello, el código es totalmente libre y estará publicado en Github.

Debido a las grandes cantidades de personas que migran todos los años a la Unión Europea, y lo susceptible que son los flujos migratorios a variables poco predecibles como las guerras, desastres naturales o epidemias, podemos concluir que siempre será un tema de interés. Por lo tanto, para mejorar lo ya desarrollado habrá que mejorar la escalabilidad del ecosistema principalmente en dos aspectos muy específicos:

- Accesibilidad al ecosistema: con esta variable nos referimos a la posibilidad de que el mayor número de personas sean capaces de utilizar el ecosistema ya desarrollado sin tener un gran conocimiento técnico de las herramientas a utilizar, consiguiendo un gran nivel de abstracción de los procesos que suceden en el *backend*. La propuesta

para lograr esta meta es realizar una interfaz gráfica implementada sobre los patrones *Adapter* desarrollados para el *backend*. Así nos aseguramos que, a pesar de cambiar las herramientas finales del ecosistema, este seguirá funcionando con normalidad.

- Rendimiento multi-hilo: en este momento el ecosistema es capaz de ejecutar un algoritmo por ejecución pero lo mejor sería que fuera capaz de ejecutar varios algoritmos sobre el mismo *dataset* en un solo flujo de ejecución, logrando así aprovechar al máximo las herramientas como Hadoop y YARN, basadas en procesamiento paralelo y distribuido, obteniendo distintos informes para cada ejecución, o integrándolos en un mismo informe.

Capítulo 5. Conclusions and future work

Migratory flows will always exist in our societies, and the origins of them have different reasons, but the common denominator is that everyone who emigrates seeks a better life than in their place of origin. There are various barriers that can hinder immigration, such as immigration policies, culture or language. These barriers generate a great extra cost in the budget for immigration and integration of the European Union, so knowing what type of people (clusters) come could significantly improve the return on investments in this aspect, since it will indicate, in a general way, some of the needs of migrants that we could be overlooking by not doing an analysis of the data.

From the beginning, this ecosystem has been planned to be flexible, easy to maintain and to add functionalities to it. That is the reason why several design patterns have been used, so that the final objects of this environment are more like plugins that can be put and removed at the convenience of the person who is going to use the ecosystem, giving him/her total freedom as flexibility. Thus, the code is totally free and will be published on Github.

Due to the large number of people who migrate to the European Union every year and the susceptibility of migratory flows to unpredictable variables such as wars, natural disasters or epidemics, we can conclude that this will always be a subject of interest. Therefore, to improve what has already been developed, it will be necessary to also improve the scalability of the ecosystem mainly in two very specific aspects:

- Accessibility to the ecosystem: with this variable we refer to the possibility that the greatest number of people are able to use the already developed ecosystem without having a great technical knowledge of the tools to be used, achieving a great level of abstraction of the processes that take place at the backend. The proposal to achieve this goal is to create a graphic interface implemented on the adapter pattern developed for the backend, thus ensuring that despite

changing the final tools of the ecosystem, it will continue to function normally.

- Multi-threaded performance: at this moment the ecosystem is able to execute one algorithm per execution, but it should be able to execute several algorithms on the same dataset in a single execution flow, in order to take advantage of the tools like Hadoop or YARN, which are based on parallel and distributed processing, managing to obtain different reports for each execution, or integrating them in the same report.

Capítulo 6. Presupuesto

Este capítulo tiene como objetivo presentar los costes estimados de desarrollo de este proyecto. Debido a que las herramientas utilizadas en este ecosistema Big Data son de código abierto y gratuitas, no tendremos costes asociados a las licencias de software.

6.1 Costes de Hardware

El requerimiento de hardware para llevar a cabo un análisis de datos dependerá principalmente de nuestras necesidades. Pero principalmente nos referiremos a volumen de datos y velocidad de procesamiento. Al no tener una excesiva cantidad de datos para analizar, y que Hadoop permite el procesamiento en local a nivel de núcleos de procesador, se ha escogido la siguiente configuración de hardware.

Tipo de Hardware	Componente Hardware	Precio
Disco Duro 1TB	SEGEATE ST1000LM35-1RK172	54,56 €
RAM	Ramaxel 8GB DDR4 2666MHz	25€
CPU	Core i7-7700HQ	320€
Placa base	Lenovo SDK0J40709 WIN	80€
Fuente de alimentación	Bateria 3 celdas 45W	70€

Tabla 6.1. Costes de Hardware

6.2 Costes de Recursos Humanos

Para el costo de la hora de trabajo, se fijará un precio de 20 € la hora debido a la diversidad de tecnologías a integrar y el valor añadido de la arquitectura utilizada.

Tarea Realizada	Horas dedicadas	Precio
Aprendizaje Mongo	30	600€
Aprendizaje Hadoop	25	500€
Aprendizaje HDFS	15	300€
Aprendizaje Yarn	20	400€
Aprendizaje Pentaho	15	300€
Aprendizaje Maven	10	200€
Desarrollo de Test	200	4000€
Desarrollo del Ecosistema	115	2300€

Tabla 6.2. Resumen de tareas y costes

6.3 Costes Totales

En este apartado mostramos el coste total de desarrolla el ecosistema:

Tipo de Gasto	Precio
Gasto en Hardware	549,56 €
Gasto en Formación	2300€
Gasto en desarrollo	6300
Coste total	9149,56€

Tabla 6.3. Resumen de costes totales

Bibliografía

[1] Causa A., Jadamba B. & Raciti F. A migration equilibrium model with uncertain data and movement costs (2017). *Decisions Econ Finan* 40, 159–175. doi.org/10.1007/s10203-017-0198-4

[2] ESA (2019), Big data for migration study - Big Data applications to boost mitigation preparedness and response to migration feasibility study business.esa.int/projects/big-data-for-migration-study

[3] EUROSTATS, ec.europa.eu/eurostat/data/database

[4] Hadoop Documentation, hadoop.apache.org

[5] HDFS Documentation, hadoop.apache.org/docs

[6] Hofleitner A., Chiraphadhanakul TV and State B. (2013) Coordinated Migration www.facebook.com/notes/facebook-data-science/coordinated-migration/10151930946453859

[7] Maven Documentation, maven.apache.org/guides/index.html

[8] MongoDB documentation, docs.mongodb.com

[9] Nagurney, A., Daniele, P. and Cappello, G. (2021), Human migration networks and policy interventions: bringing population distributions in line with system optimization. *International Transactions in Operational Research*, 28: 5-26. doi:10.1111/itor.12815

[10] Pentaho Documentation, help.pentaho.com/Documentation/8.2/

[11] Pentaho Report, subscription.packtpub.com/

[12] Yarn Documentation, hadoop.apache.org/docs

[13] Zagheni E., Garimella V.R.K., Weber I. and State B. (2014) Inferring International and Internal Migration Patterns from Twitter Data. WWW Companion 439-444. ingmarweber.de/wp-content/uploads/2014/02/Inferring-International-and-Internal-Migration-Patterns-from-Twitter-Data.pdf