Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Implementación de un entorno de almacenamiento masivo en la nube con el propósito de análisis e informes.

Autor: Alejandro Vázquez Gómez

Tutor/es: Iván González Diego

2021

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado
Implementación de un entorno de almacenamiento masivo en la nube con el propósito de análisis e informes

Autor: Alejandro Ramón Vázquez Gómez

Tutor/es: Iván González Diego

TRIBUNAL:

Presidente:

Vocal 1º:

Vocal 2º:

FECHA:

Índice

Índice de tablas

Índice de figuras

Índice de código

1. Introducción.

1.1. Resumen.

En este trabajo se presenta y recoge el uso conjunto de varias de las áreas más relevantes en la informática actual, como la Nube, el Big Data o la Ciencia de datos, con el propósito de realizar en la nube la implementación de un entorno de almacenamiento masivo capaz de realizar análisis e informes y que sirva como caso de ejemplo para modelizar la virtualización o digitalización de entornos tradicionales.

Comenzaremos dando un breve repaso a los fundamentos y a la situación en la que se encuentran las tecnologías a emplear para posteriormente elegir un conjunto de datos y a partir de este el conocimiento que queremos extraer, los componentes del sistema adecuados, la arquitectura necesaria para comunicarlos y por último la propia implementación del sistema junto a su documentación correspondiente.

Palabras clave: Almacenamiento masivo, Nube, Ciencia de datos, Azure.

1.2. Abstract.

This Bachelor dissertation presents and elaborates on the joint use of several of the relevant fields in today's computing, such as the Cloud, Big Data or Data science, in order to create a massive, cloud-based, storage environment capable of analysis and reporting that also serves as proof of concept to model the virtualization and/or digitization of traditional environments.

We will start by giving a brief review of the fundamentals and the state of the art of the technologies to be used, then choose a data set and from there the knowledge we want to extract from the data, the appropriate system components, the architecture necessary to communicate them and finally the implementation of the system together with its corresponding documentation.

Keywords: Big Data, Cloud Computing, Data science, Azure.

1.3. Resumen Extendido.

En este trabajo haremos uso de la Nube, conocida como Cloud Computing o solamente Cloud y entendida como el uso de aplicaciones y servicios ofertados a través de Internet; Big data, que son datos con gran variedad de formato y presentes en volúmenes que incrementan con cada vez mayor velocidad, según la definición clásica de las 3 Vs y por último la Minería de datos, que nos permite descubrir anomalías, y sacar conclusiones, en base a distintas técnicas y métricas computacionales y estadísticas. Estas tecnologías resultan inmensamente importantes en la actualidad hasta el punto de que IBM estima que 2.5 quintillones (2,500,000,000,000,000,000) de bytes son creados, o que cada persona usa de media 36 servicios con base en la nube cada día. [20]

Para poder llevar a cabo lo explicado en el apartado anterior será necesario primero elegir los datos sobre los cuales trabajaremos y plantear la información que deseamos conocer, esto irá acompañado de un estudio de la base teórica y de los proyectos de relevancia para cada campo de interés en nuestro proyecto; una vez teniendo claro nuestro marco podremos determinar las herramientas,infraestructura (al estar situado en la Nube) y estructura de nuestro sistema e implementarlo. Veremos que es necesario seguir un proceso estructurado de desarrollo y aplicar concienciudamente los preceptos de la Ingeniería del Software en cada etapa de este proceso, pues una de las principales dificultades de este trabajo va a consistir en elegir entre las múltiples alternativas que aparecerán a cada paso del proyecto; de lo contrario dada la enormidad de los campos con los que trabajamos existe un gran riesgo de incurrir en graves errores de especificación y concepto.

En cuanto a la motivación personal para realizar un trabajo sobre esta temática, tratamos aquí algunas de las áreas más interesantes, activas y con más aplicaciones en la informatica actual y sin embargo apenas se imparten en la carrera. Vemos pues que suelen enseñarse de forma aislada cuando en los desarrollos empresariales se emplean de forma conjunta entre ellas o con otro tipo de tecnologías y además se extiende a buena parte de los TFGs existentes en temas de Big Data y Cloud: Se centran en un único aspecto de la tecnología que estudian o bien presentan un componente teórico tan fuerte que se ve limitada su aplicabilidad en relación con lo visto en el mundo profesional.

Además nos gustaría hacer mención a la metodología de desarrollo y al proceso de trabajo seguido en general pues la mayoría del trabajo ha tenido lugar durante el confinamiento por el Covid-19 o en los meses de verano, ante esta situación hemos optado por trabajar mediante videoconferencias, principalmente a través de Teams. Antes de pasar al trabajo propiamente

dicho, recogemos aquí una breve descripción de los objetivos y contenido de cada apartado contenido en este trabajo:

- La primera sección consiste en los prolegómenos al trabajo y consta de los resúmenes en castellano e inglés así como de esta introducción.
- La segunda sección tratará acerca de la elección de un conjunto de datos y las cuestiones relacionadas con dicha elección: la estructura que presentan los datos, el tipo de búsquedas a realizar, el conocimiento que se quiere extraer y las posibles implicaciones de cara a la arquitectura del sistema.
- La tercera sección se corresponde con el estudio de las herramientas existentes en los distintos campos que emplea este proyecto junto con una sucinta explicación de los fundamentos teóricos de su funcionamiento.
- La cuarta sección contiene la descripción de la arquitectura elegida, la elección de los componentes adecuados entre los expuestos en el apartado anterior, y por último el proceso de implementación del proyecto, desde la carga de datos a la interfaz gráfica.
- En la quinta sección encontraremos expuestas las pruebas necesarias para validar nuestro sistema, varios casos de uso para ejemplificar su funcionamiento y una breve descripción o manual de cómo usar el sistema.
- Por último en las secciones sexta y séptima recogeremos las conclusiones fruto de la realización de los apartados anteriores así como un presupuesto aproximado, y la bibliografía con la documentación empleada a lo largo del proyecto.

De esta manera puede dividirse nuestro trabajo en tres partes, una primera de exploración de los conceptos y herramientas existentes, una segunda de implementación y documentación del proceso seguido y por último una tercera donde se recogerán las conclusiones pertinentes extraídas a lo largo de la elaboración de este trabajo.

2. Elección de los Datos

2.1. Conjuntos de Datos.

En primer lugar vemos necesaria una explicación acerca de la estructura de este apartado, donde a primera vista parece más razonable incluir y explicar únicamente los datos que van a usarse, sin embargo, al empezar este proyecto no partíamos de datos ya determinados sino que se ha llevado a cabo un proceso de selección a través del cual se han identificado las características de diversos conjuntos de datos, su adecuación para el proyecto y su influencia en el resto de apartados; además, como veremos al final de este apartado, en base a consideraciones de arquitectura, facilidad de implementación e interés propio hemos tratado de considerar dos conjunto de datos para dos desarrollos distintos. Para poder definir con precisión la arquitectura, las herramientras y las características de la extracción de conocimiento que queremos emplear, es preciso conocer primero los datos que va a cargar el sistema, los tipos de datos a los que pertenecen, la información que nos es posible conocer y la que nos interesa, etc. Si en el caso contrario se fija primero la arquitectura y composición del sistema para buscar datos que encajen con nuestro sistema después, dificultamos enormemente la integración y funcionamiento de los distintos componentes del sistema, por tanto esta idea aparecerá repetida varias veces a lo largo del trabajo.

En cuanto a los propios datos, se han obtenido a través de Kaggle, una comunidad centrada en el intercambio de conjuntos de datos (entre otras funcionalidades puede compartirse también código sobre dichos conjuntos) y compuesta por personas de todo el mundo con intereses en la Ciencia de Datos y Machine Learning. Acerca de la misma podemos decir que cuenta con decenas de miles de conjuntos de datos los cuales pueden ser filtrados y ordenados por distintos parámetros tales como peso de los archivos, valoración en términos de 'likes' o usabilidad, etc. Existen otras maneras de obtener datos pero habiendo trabajado previamente con Kaggle en la carrera y siendo uno de los objetivos de este trabajo el estudio de la virtualización nos ha parecido apropiado hacer uso de Kaggle también para este trabajo.

A continuación definiremos los criterios según los cuales valoraremos la idoneidad de nuestros datos:

- Tamaño: Que deberá ser adecuado como para resultar manejable pero también tener sentido su almacenamiento en un proyecto de estas características; el Big Data suele medirse en términos de Terabytes pero para una prueba de concepto consideramos conjuntos de datos con un tamaño de 750 MegaBytes o más como suficientes, si bien de ser posible intentaremos trabajar con tamaños mayores.
- Usabilidad: Kaggle define la usabilidad según un conjunto de datos esté bien documentado, sus formatos sean legibles con facilidad y exista código de ejemplo para ilustrar como se trabaja con esos datos. Es decir nos interesa que el conjunto presente documentación legible y relevante, sea fácil de manejar en cuanto a formato y estructura de los datos y podamos hacernos cierta idea de cómo trabajar con él.
- Relevancia de la información: El conocimiento que se pueda extraer de los datos debe resultarnos lo suficientemente interesante como para luego poder aplicarle minería de datos y presentarlo en un dashboard, es decir, no nos sirve aplicar algoritmos solamente a unos datos en un formato presentable, debe de ser posible interpretar la información obtenida para que apoye análisis e hipótesis sobre los datos.

Por tanto, hemos seleccionado 3 alternativas que de acuerdo a los criterios descritos anteriormente resultan de interés y que analizaremos para ver cual resulta más idónea para el proyecto.

2.1.1. Actividad de teléfonos móviles en las ciudades de Italia.

2.1.1.1 Descripción del conjunto.

Partiendo de la descripción que se hace en Kaggle del conjunto de datos podemos hacernos una idea adecuada de la funcionalidad y el contenido del mismo por lo que nos limitaremos a seguir en líneas generales lo expuesto en el enlace. Este conjunto de datos fue creado para una competición italiana en Big Data en el 2014 y se compone de los registros obtenidos durante una semana de actividad en los Call Detail Record (CDR) de la ciudad de Milán y la provincia de Trentino-Alto Adigio/Tirol del Sur [1].

Generalmente, cada vez que un usuario realiza una interacción a través de una red de telecomunicaciones, el operador asigna una Base Station (BSR), la terminología varía según fabricantes y este es el término más extendido, que entrega la comunicación a través de la red. Luego, se crea un nuevo CDR que registra el tiempo de la interacción y la Base Station que la maneja.

Junto a lo anterior es muy importante mencionar que los datos que aparecen en Kaggle forman un subconjunto relativamente pequeño del total, pues trabajan solamente sobre actividad de una semana en Milán, no incluyen los datos de Trentino; el original, publicado en Nature, recoge además datos sobre clima o redes sociales (Twitter), siendo este último de especial interés para nuestro estudio por contener los tweets geolocalizados producidos en Milán y Trentino-Alto Adigio.

Esto presenta la complejidad adicional de tener que decidirnos por una de las dos posibles versiones del conjunto; es decir, al tener la versión de Kaggle un menor tamaño resulta más facil trabajar con ella en la nube sin embargo no cuenta con los datos de Trentino. El Big Data fue ideado para el procesamiento de grandes volúmenes de datos y es ahí donde destacan sus ventajas sobre las Bases de Datos Relacionales tradicionales. Ante esta situación se ha optado por tratar de realizar las primeras fases de prueba e implementación con el conjunto que aparece en Kaggle y eventualmente cuando el proyecto esté lo suficientemente avanzadO, cargar el conjunto de datos extendido a nuestra Nube con el entorno ya implementado. En cuanto a las fuentes y datos contenidos el conjunto recoge CDRs en base a las siguientes actividades:

- Mensajes recibidos.
- Mensajes enviados.
- Llamadas recibidas.
- Llamadas enviadas.
- Actividad en internet.

Es necesaria también una aclaración porque toda la actividad en internet generada a través de un móvil implica un CDR tanto al comenzar o terminar una conexión a internet, pero únicamente si esta dura más de 15 minutos o se intercambian más de 5 MegaBytes.

Por último, resulta necesario explicar con cierto detalle el formato de los datos presentes en el conjunto, este se compone de archivos en formato CSV, aunque la información geográfica se encuentra en formato GeoJSON, una extensión del formato JSON relacionada con otros como TopoJSON, más concretamente podemos observar que los ficheros CSV principalmente presentan campos numéricos con alguno de tipo Date y Text/Varchar, además podemos observar que existe una gran cantidad de nulos en los campos numéricos debida a las ocasiones en las que no se registra actividad en una

celda; en cuanto a los GeoJSON, en éstos se encuentra codificada información geográfica de las provincias italianas y una distribución en celdas del área de Milán y del Trentino, de esta manera podemos analizar la actividad que ha registrado la red de telecomunicaciones en cada celda del GeoJSON.

2.1.1.2 Conclusiones y análisis a partir del conjunto.

Según lo expuesto arriba puede verse que de estos datos puede extraerse gran cantidad de información y realizar inferencias acerca de cuestiones tales como patrones de mobilidad, detección de aglomeraciones, densidad de población o posibles ubicaciones de las antenas de telefonía (mediante clusterización), además nos es posible enriquecer los datos ya existentes como los del Instituto nacional italiano de estadística (ISTAT) o los distintos campos del conjunto de datos ampliado. Nuestra intención es poder analizar cuestiones tales como el grado de penetración del móvil en Italia, diferencias en hábitos y mobilidad entre una ciudad global como Milán y una región relativamente poco poblada y rural como es el caso del Trentino-Alto Adigio, etc. Ciñéndonos en la medida de lo posible a los datos de telecomunicaciones pero complementándolos donde sea preciso.

Estas cuestiones por lo general se acercan más al dominio de la sociología pero sin embargo constituyen un interesante ejemplo de como el Big Data y la Minería de datos se aplican a disciplinas que pueden verse beneficiadas por el análisis informático, siendo este un intercambio que cada vez se da con mayor frecuencia y cada vez produce más resultados; en este caso para inferir patrones y dinámicas de comportamiento a través del uso conjunto de varias herramientas y disciplinas.

En cuanto al cumplimiento de los criterios definidos arriba, vemos que tiene tanto un tamaño manejable en el subconjunto reducido como uno muy adecuado para emplear Big Data en el conjunto original. Presenta además una documentación muy extensa y si bien en la legibilidad de los datos al no haber trabajado antes con GeoJSON resulta un poco extraña no es para nada mala y por lo general es un muy buen candidato para usarlo de base en nuestro sistema.

2.1.2. Datos acerca de la actividad de Airbnb en Madrid.

2.1.2.1 Descripción del conjunto.

En el caso de este conjunto, en Kaggle no viene adjunta ninguna descripción del mismo aunque vienen del proyecto sobre Airbnb de Murray Cox [3] por lo que toda la información viene inferida por observación, aunque si bien no es muy difícil entender el contenido del conjunto y dar sentido a esos datos, esa pérdida de contexto dificultará luego un posible análisis. Esto no quiere decir sin embargo que los datos no resulten usables o interesantes, todo lo contrario, por ejemplo Madrid actualmente presenta una enorme proliferación de pisos de alquiler para turistas y aplicar minería de datos puede resultarnos enormemente esclarecedor para llegar a conclusiones que nos permitan entender no sólo esta situación sino también otras como la distribución y evolución de los precios en estas plataformas como Airbnb, Booking o Vrbo.

Antes de entrar a describir el conjunto como tal es preciso explicar un poco de terminología propia de este tipo de negocio. Los listing representan las casas que los hosts o huéspedes, es decir, los propietarios, listan en el portal de Airbnb como disponibles para ser alquiladas, scraping se refiere a la práctica de extraer información de la web de Airbnb para organizarla en forma de tabla o Base de Datos. Una vez tenemos una idea general del contexto de estos datos resulta posible ver entonces que presentan un grado de semejanza considerable (Datos numéricos, información geográfica sobre una ciudad, etc.) con el conjunto de datos anterior en cuanto a estructura, formato y tipo de datos, por lo que el grado de detalle empleado en este apartado será algo menor, pues ya tenemos una alternativa similar y mejor documentada arriba, pero por lo general sigue la estructura general de uno o varios archivos GeoJSON con información geográfica, en este caso de los barrios y distritos de Madrid, junto a información cronológica de las reservas, barrios y reviews en formato CSV.

2.1.2.2 Conclusiones y análisis a partir del conjunto.

En primer lugar y como ya adelantábamos antes, existe bastante similaridad entre este conjunto de datos y el primero hasta el punto de que si bien ambos resultan muy interesantes por las posibilidades que ofrecen a la hora de analizarlos y extraer conocimientos mediante minería de datos, se abre la problemática de elegir entre uno y otro por documentación o usabilidad en vez de por funcionalidad y estructura, lo que acorta nuestras posibilidades de elección en temas de arquitectura o componentes y empobrece un poco el

estudio que estamos llevando a cabo. En parte por ello el siguiente conjunto de datos presenta una estructura bastante distinta a los anteriores.

2.1.3. Extractos de las entrevistas que han tenido lugar en la NPR (Radio pública estadounidense).

2.1.3.1 Descripción del conjunto.

Si bien la descripción que aparece en Kaggle de este conjunto es más escueta que la de la actividad móvil en Italia, viene bastante más información que en el caso de los datos de Airbnb y entre lo ya recogido ahí y el contexto aportado por cuenta propia será posible hacerse una mejor idea de la estructura de dicho conjunto junto a su significancia y los pros y contras de trabajar con él. Este conjunto de datos alberga las transcripciones de distintos programas de la National Public Radio, más conocida por sus siglas como NPR, a lo largo de 20 años, dando lugar a más de 140.000 transcripciones y en torno a 10.000 horas de audio [2].

La NPR lleva operando desde hace 50 años y si bien no es la radio más extendida o escuchada del país (Premiere Networks tiene ese mérito) está considerada como el medio de comunicación más fiable de Estados Unidos y siendo este el motivo principal por el que hemos elegido el conjunto de datos, presenta además una enorme variedad temática, desde los programas musicales como los Tiny Desk Concerts o los distintos podcasts sobre temas igualmente diversos a los dos programas más destacados: Morning Edition y All Things Considered, ambos con 14'6 millones de oyentes semanales y de audiencia, muy cerca de ser los programas más escuchados de la radio norteamericana; aunque dicha audiencia presenta un cierto sesgo a ser graduada universitaria, dada esta enorme variedad estos datos resultan muy interesantes de cara a extraer conocimiento de sus programas pues no presentan sesgo político, los temas candentes y la cultura de Estados Unidos, en lo cual entraremos posteriormente. Cabe añadir además que la importancia relativa ha descendido bastante con la aparición de nuevas tecnologías de telecomunicaciones por lo que el número de oyentes también se ve afectado, pues en los años 30 hasta su colapso por la aparición ya mencionada de la televisión y posteriormente internet, se podían ver programas con 30 o 20 millones de espectadores; sin embargo, con la radio por satélite se ha visto un cierto repunte y la radio sigue siendo una representante tan válida como siempre de las preocupaciones y la actualidad social norteamericana.

En cuanto a la estructura del conjunto de datos propiamente dicho, esta es similar a la de los conjuntos anteriores (CSV y JSON) aunque presenta

ligeros cambios:

- Los metadatos (información sobre fechas y nombre de programa) se encuentran en episodes.csv, la información sobre los presentadores en host-map.json y host-id.json.
- En utterances-2sp.csv aparece recogida la información sobre las conversaciones entre dos interlocutores (para tener en cuenta diálogos) y en utterances se encuentran las sentencias, es decir, la propia conversación.
- En split-ns2.json se encuentra una división en tres conjuntos de entrenamiento/validación/test para el uso de redes neuronales, aún cuando este trabajo no haga uso del Machine Learning es interesante encontrarnos esto pues representa prueba de la utilidad de la información.

Principalmente trabajaremos sobre los archivos episodes.csv y utterances.csv pues contienen las transcripciones propiamente dichas y su contexto (Año, programa, etc.), pues cuando veamos la aparición de un cierto tema deberá ser tenido en cuenta donde y cuando aparece para juzgar su grado de influencia.

2.1.3.2 Conclusiones y análisis a partir del conjunto.

Tal y como se comentaba anteriormente la intención es, a grandes rasgos, analizar la influencia de ciertas corrientes culturales y de determinados acontecimientos y temas sociales señalados en los medios de comunicación y en la información que se crea cada día en la sociedad norteamericana (Tomemos como ejemplo el auge de nuevos estilos musicales, la guerra de Irak, el Brexit o las recientes elecciones), es decir, comprobar hasta qué punto aparecen ciertos aspectos polémicos o relevantes en la radio y a través de ello tratar de hacernos una idea de su impacto en la vida cotidiana de Estados Unidos. En esto toma relevancia la ya mencionada enorme variedad temática de la NPR y su ausencia de sesgo político, que podría entorpecer nuestro análisis. No se trata por tanto de un análisis sociopolítico sino de curiosidad por nuestra parte en cuales de los temas de interés han influido más en la vida pública estadounidense.

Nuestra primera intención para este conjunto es emplear motores de búsqueda, es decir, indexado, por múltiples motivos: primero por la idoneidad de hacer búsqueda textual sobre índices, segundo por la viabilidad, como veremos adelante, resulta posible buscar sobre varios índices, importar e indexar ficheros CSVs en motores de indexado como Lucene además de mostrar una

propuesta diferente, pues con la Nube suelen existir varias soluciones ya implementadas para un mismo problema y dado este conjunto también puede trabajarse con él de forma similar a la señalada en el primer conjunto.

En cuanto a los criterios de valoración vemos que la validez de este último conjunto es también considerable, pues aunque no sea tan flexible en tamaño como el primer conjunto, es también razonablemente extenso (20 años de transcripciones), la información contenida es de utilidad e interés, la documentación aportada es también satisfactoria y la estructura de los datos deja poco lugar a dudas.

2.2. Elección de los datos.

Una vez se han descrito con un nivel de detalle suficiente los conjuntos de datos candidatos es posible entrar a valorarlos y tomar una decisión en base a la cual continuar nuestro proyecto, pues el conjunto de datos influirá en la arquitectura general de todo el sistema y por tanto en el desarrollo posterior de gran parte de nuestro trabajo.

En vista de lo expuesto arriba puede verse que el primer y el segundo conjunto de datos (Actividad móvil en Italia y Datos Airbnb en Madrid) van a proporcionar búsquedas sobre campos numéricos, de tipo fecha o de tipo cadena/string de estructura más o menos similar a las Bases de Datos relacionales, por lo que podremos interactuar con los datos de manera parecida a como haríamos con SQL mientras que el back-end, donde se encuentra realmente el Big Data, quedará relativamente escondido, por la contra en el tercer conjunto emplear un esquema semejante al relacional/SQL no resultará tan eficiente al componerse de los extractos de conversaciones y será más adecuado realizar búsquedas textuales mediante motores de búsqueda como los basados en Lucene (Indexado).

Si tenemos en cuenta el tamaño del segundo conjunto de datos, su similitud en líneas generales con el primero y su falta de documentación además de la facilidad de implementar un proyecto en la nube. Finalmente nos hemos decantado por ignorar dicho conjunto para tratar de implementar conjuntamente dos desarrollos, un desarrollo ”principal” con los datos del primer conjunto en base a una solución con clusters de datos, minería de datos, dashboards, etc. y otro centrado en motores de búsqueda como Elasticsearch (Y su implementación en la nube) con el tercer conjunto; ambos esquemas serán revisados más adelante en el punto 4.1. Por último a modo de resumen de este apartado incluimos la siguiente tabla:

Conjunto	Tamaño	Usabilidad y Documentación	Observaciones
Actividad móvil Italia	De 1GB en Kaggle a varios GB el original.	[1]	Conjunto elegido para el desarrollo principal del trabajo.
Airbnb Madrid	613MB	[3]	Descartado debido a su similitud con el conjunto anterior y falta de documentación.
Entrevistas NPR	832MB	[2]	Considerado para un desarrollo secundario en caso de ser viable.

Tabla 1: Conjuntos de Datos

2.3. Validación de los Datos.

Cabe realizar aquí una digresión pues conforme vayamos analizando los datos puede resultar adecuado realizar un tratamiento de validación de los datos con el fin de eliminar los valores atípicos y garantizar la validez y la robustez de nuestros análisis posteriores.

En estadística un valor atípico u outlier es aquel que se encuentra extremadamente distante del resto de observaciones desde un punto de vista numérico (La distancia la determina el analista con arreglo al propio conjunto de datos). Puede ser necesario además tener un cuidado especial con estos valores pues pueden interferir fuertemente con nuestros procesos y dar lugar a resultados incorrectos, por ejemplo si analizando los datos de la actividad móvil vemos que de media las celdas presentan en torno a 100 llamadas entrantes pero en otra hay 3000 tendremos un valor atípico; en conjuntos de datos extremadamente grandes como los que trataremos es habitual encontrarse outliers y es raro descartarlos automáticamente al producirse ”naturalmente”, por ejemplo en el centro de Milán esperamos encontrar más llamadas que en las afueras, sin embargo en nuestros procedimientos al trabajar con distancias considerablemente grandes puede resultar que no tengamos que descartar datos verdaderamente atípicos.

A su vez existen multitud de métodos para detectar los valores atípicos, algunos más conocidos que otros pero por lo general todos con una excelente documentación como la distancia de Mahalanobis, el algoritmo K-Vecinos o el error estándar de los residuos. Deberían sin embargo tenerse también en cuenta los requerimentos de memoria de nuestros métodos dado el tamaño de

los conjuntos de datos, es decir, al ser tamaños del órden de Gigabytes puede no ser posible introducir todos los datos con los que trabaja el algoritmo en memoria a la primera y por tanto consumir considerablemente más tiempo de ejecución, siendo esto de especial importancia en la nube donde se cobra por tiempo de ejecución en base a tarifa según el poder computacional alquilado.

Finalmente en el tratamiento de los datos previo a la implementación definitiva se ha visto que no era necesaria esta técnica en vista de la propia estructura de los datos, pero uno de los algoritmos explorados para analizar el primer conjunto de datos, DBSCAN, puede emplearse también para la detección de valores anómalos aunque sea un algoritmo de clusterización, por lo que recogeremos más adelante el resultado obtenido.

3. Estado del Arte.

Este apartado lleva a cabo un repaso de las herramientas y proyectos de referencia en los distintos apartados que conformarán los componentes de nuestro sistema; nos centraremos en explicar la funcionalidad concreta de cada proyecto y sus diferencias entre ellos si bien entraremos puntualmente en algunos conceptos teóricos, pues un estudio concienzudo de los fundamentos de algunos de los proyectos aquí descritos puede constituir un TFG por sí mismo.

De esta manera, similar al caso anterior con los conjuntos de datos, una vez tengamos una idea del funcionamiento y funcionalidad de cada herramienta podremos determinar cuáles de los proyectos aquí descritos serán adecuados cuando establezcamos la arquitectura de nuestro entorno, pues si bien es posible decidir a priori los componentes con los que queremos trabajar, puede que luego no sea posible establecer una comunicación como tal entre ellos o resulte complicado realizar con ellos las tareas que sean necesarias.

3.1. Almacenamiento Masivo.

A la hora de hablar de Big Data, resulta necesario entender que en él coexisten una gran cantidad de proyectos, cada uno con su arquitectura y filosofía particulares y unidos por el hilo común de salirse del estándar de las bases de datos relacionales tradicionales, estamos pues ante un campo bastante más heterogéneo de lo que pudiera parecer aunque como adelantábamos en parte en el apartado 2.2 y veremos posteriormente, existen interfaces que permiten comunicarnos con los programas que veremos mediante lenguajes de Manipulación de datos semejantes a SQL o a través de lenguajes de programación, mediante APIs JDBC/ODBC o una integración directa. [14] [13]

Podemos definir ahora de una manera un poco más técnica el Big Data: Consiste en el almacenamiento y procesamiento masivo de datos no estructurados, cuyo tamaño impide el uso de los enfoques tradicionales. Por si mismo esto no nos dice mucho pero si tenemos en cuenta que los costes de almacenamiento y computación disminuyen con la aparición de nuevas tecnologías, resulta posible almacenar volúmenes de datos previamente considerados inmanejables en esquemas que no requieren de la rigidez de los sistemas relacionales y una mayor escalabilidad de cara a aumentar los datos contenidos.

Dentro de nuestro estudio al no poder abarcar la completa extensión de

este campo nos centraremos en Hadoop, una plataforma de almacenamiento áltamente escalable, Spark, un motor de análisis para el Big Data y Machine Learning y Mongo/CosmoDB, una base de datos NoSQL. De esta manera veremos tanto sistemas de archivos distribuidos como bases de datos no relacionales y podremos comprobar las características, pros y contras de cada enfoque.

3.1.1. Hadoop.

Hadoop consiste en un framework, o conjunto de utilidades y librerías, con el objetivo de proporcionar almacenamiento y procesamiento de grandes cantidades de datos a través de modelos de programación relativamente sencillos. Estos datos se almacenan en cientos o miles de nodos de cálculo, proporcionando así una enorme capacidad de escalabilidad y eficiencia. Hadoop cuenta con un conjunto de utilidades básicas a las que posteriormente se agregan paquetes con diversa funcionalidad, estos módulos básicos son: [5] [6]

- Common contiene las librerías Java y utilidades que emplearán después otros módulos.
- HDFS, como indican sus siglas (Hadoop Distributed File System), es un sistema de archivos distribuido que permite almacenar datos entre múltiples máquinas. HDFS está diseñado para dar una alta redundancia, disponibilidad y escalabilidad por lo que HDFS replica sus nodos en dos servidores (de base), dichos nodos pueden ser de dos tipos: Maestros o NameNode, que almacenan metadatos como la Id del bloque o su localización y dirigen a los esclavos o DataNode, que son los que almacenan propiamente los datos, divididos a su vez en bloques.
- YARN se encarga de planificar los trabajos y de dividir los recursos entre un Resource Manager global ayudado por un NodeManager en cada máquina y varios ApplicationMaster.
- MapReduce es una implementación del modelo de programación del mismo nombre: Map es una función aplicada a cada elemento de la entrada de datos, como parámetros recibe un par clave/valor y su salida es una lista de pares, dichos pares se agruparán luego de entre las listas resultantes por cada clave existente, Reduce tiene como entrada cada clave única de la salida de Map y una lista de los valores asociados a esa la clave para realizar operaciones de agregación que devolverán un valor o lista con el resultado.

Vemos entonces que en cuanto a funcionalidad y utilidad Hadoop resulta muy eficiente para almacenar datos sin preocuparnos por tamaño, disponibilidad o formato, junto a ello Hadoop resulta ideal para integrar otras utilidades como HBase/Phonenix o Hive que nos evitan tener que interactuar con las complejidades de su funcionamiento o añaden sobre la funcionalidad que ya nos aporta Hadoop para solventar problemáticas concretas, por tanto puede decirse que Hadoop constituye parte de los cimientos sobre los que luego se construye un sistema completo. [6]

3.1.2. Spark.

En directa relación con el apartado anterior Spark es un sistema de computación distribuída construído en base al modelo MapReduce, de hecho puede decirse que Hadoop es un medio para implementar Spark pues Spark puede interactuar con una variedad de sistemas de almacenamiento y bases de datos como Cassandra o Amazon S3. Sobre esta base Spark añade todavía más funcionalidad como el procesamiento en Batch (Lotes) y permite un enorme aumento del rendimiento, debido a un uso extensivo de la memoria frente a la predominacia del disco externo en Hadoop. [8] [7]

Spark se suele dividir a su vez en los siguientes componentes: [8]

- Apache Spark Core es el motor de ejecución que sirve de base a la plataforma y proporciona la computación en memoria (In-memory computing) y referencias a los datos en los sistemas de almacenamiento externos.
- SparkSQL proporciona soporte a SQL a través de la implementación del mismo, cabe hacer aquí una digresión y explicar que Spark tiene tres API's para interactuar con los datos: RDD, DataFrame y Dataset, RDD viene de serie y el soporte a las otras dos API's lo realiza SparkSQl, además del soporte a interfaces de tipo ODBC/JDBC y consola.
- SparkStreaming emplea la capacidad de planificación de Spark para trabajar con Datasets Distribuidos Resistentes (RDD), conjuntos de datos con tolerancia a erroresy paralelizables sobre datos recibidos en lote/batch.
- Mlib es un framework distribuido altamente eficiente que proporciona implementación a muchos de los algoritmos de Machine Learning y Minería de datos más comunes

- GraphX es un framework pensado para el procesamiento de grafos y proporciona dos API's para tal propósito, una basada en MapReduce y otra basada en Pregel.
- Existen además otros componentes que sin ser parte de la 'base' de Spark suelen considerarse relevantes, tales como Tachyon o Spark R.

Entre las ventajas principales de Spark encontramos pues su considerable rendimiento, el soporte a una enorme variedad de lenguajes (Java, Scala, Python...) y el soporte a consultas interactivas (SQL) y la facilidad de uso que ello conlleva, Machine Learning y a algoritmos con base en la teoría de Grafos aunque históricamente en materia de seguridad han existido alternativas mejores. [7]

Si bien Spark supone un considerable paso adelante en las capacidades de Hadoop en cuanto a procesamiento, vemos que en realidad se complementan, pues Spark presenta una enorme capacidad de procesamiento gracias al uso que hace de la memoria y Hadoop presenta una mayor seguridad y un coste muy bajo de operación, por lo que se complementan perfectamente trabajando juntos aunque también puedan hacerlo por separado.

3.2. Indexado.

Si bien este es el apartado más corto de los que componen esta sección, los motores de indexado no dejan de suponer una tecnología rápida, flexible y fiable para el manejo y análisis de datos, aunque también es un campo un tanto monolítico pues los dos proyectos más representativos: Solr y Elasticsearch se implementan ambos sobre la API Apache Lucene. Por tanto únicamente hablaremos de Elasticsearch, el más extendido de los dos y el que más facilidades presenta para implantar en la nube.

3.2.1. Elasticsearch.

Elasticsearch es un motor de búsqueda, indexado y analítica distribuída que consiste en una implementación de Apache Lucene, una librería de código abierto pensada para dar funcionalidad a motores de búsqueda y proporciona su servicio prácticamente en tiempo real para todo tipo de datos, además, su funcionamiento soporta el uso de RESTful (Implementación de la arquitectura REST) y por tanto se pueden emplear peticiones HTTP. Otro de los puntos fuertes de Elasticsearch es su flexibilidad pues es capaz de operar en una gran variedad de casos de uso, entre ellos está el manejo y análisis de

coordenadas geoespaciales, lo que resulta de interés dadas las características de nuestros datos. [12]

En cuanto al funcionamiento interno de Elasticsearch puede describirse en base a su terminología básica, pues guarda cierto parecido con el esquema de un SGBD y pueden trazarse analogías que nos ayuden a comprenderlo: [12]

- Nodo: Una instancia en ejecución de Elasticsearch, según las capacidades de cada máquina puede haber varias en ejecución en un mismo servidor.
- Clúster: Colección de uno o más nodos
- Índice: Colección de uno o más documentos, dividida a su vez horizontalmente en fragmentos, los cuales tienen las propiedades del documento pero menos objetos de manera que cada fragmento es independiente pero una réplica del original.
- Documento: Un documento es una colección de campos en formato JSON con un identificador único.

De esta manera y simplificando bastante puede entenderse un índice como una tabla en una Base de Datos Relacional, siendo la Base de Datos el clúster y cada fila un documento.

En último lugar, debe mencionarse que Elasticsearch no es un producto aislado sino que forma parte de y se implementa generalmente en un stack conocido por el acrónimo ELK y compuesto por los siguientes elementos: [12]

- Elasticsearch propiamente dicho.
- Kibana, un dashboard dedicado a visualización y antecesor de Grafana (Kibana funciona exclusivamente para Elasticsearch mientras que Grafana puede funcionar con un amplio conjunto de herramientas).
- Beats, una plataforma para el manejo de agentes para la recopilación de datos de diversos tipos tales como logs, métricas, datos de red, etc.
- Logstash, que proporciona una tubería para el procesamiento de logs y otro tipo de datos para cargarlos en Elasticsearch.

3.3. Minería de Datos.

A la hora de definir la Minería de Datos se la puede considerar como un campo de la estadística computacional centrado en el análisis de grandes cantidades de información, contenida en conjuntos de datos, con el objetivo de descubrir patrones y relaciones entre esos datos. Es un área nacida de la colaboración entre varias disciplinas (Inteligencia Artificial y Machine Learning, Estadística y Bases de Datos) pues combina algoritmos capaces de aprendizaje, tanto supervisado (aquel que trabaja con datos 'etiquetados') como no supervisado (que trabaja con datos 'no etiquetados'), métricas y técnicas estadísticas con el estudio del almacenamiento e indexado de los datos [21].

Esta disciplina se encuentra dividida en tres modelos diferentes: Aprendizaje supervisado (K-Vecinos por ejemplo) donde se aprenden funciones, relaciones que asocian entradas con salidas, aprendizaje no supervisado (K-medias), en los que no estamos interesados en ajustar pares entrada-salida, sino en aumentar el conocimiento estructural de los datos disponibles y Aprendizaje por refuerzo, cuyo objetivo es maximizar la recompensa que obtiene un agente software por actuar, de manera similar a un agente racional en términos de IA. En cuanto a las técnicas, que no algoritmos, más utilizadas destacamos las siguientes:

- Asociación, que nos permite descubrir relaciones entre distintos variables/elementos de un conjunto de datos empleando una serie de métricas: Soporte, confianza y lift.
- Clasificación, donde se agrupan distintos datos en varias categorías y Regresión, que describe la aparición de una variable según la presecia de otras.
- Detección de Anómalos, ya descrita anteriormente.
- Clustering, o clusterización, divide los datos en grupos también, pero a medida que analiza los datos y descubre patrones en vez de determinarlos a priori.

Expondremos ahora los dos lenguajes que más popularidad y extensión tienen en la comunidad que se dedica al estudio de esta disciplina, R y Python.

3.3.1. R

R es un lenguaje de programación interpretado orientado al análisis estadístico, representación gráfica y reporting y fue concebido en los años 90 como un desarrollo del anterior lenguaje S con influencias de Scheme, un dialecto de Lisp, si bien se encuentra dirigido a un nicho de mercado muy concreto goza de una gran aceptación hasta el punto de estar en el momento de ser escrito este trabajo en la octava posición del índice TIOBE. R es además Software Libre y aunque puede trabajarse con el desde su propia consola, cuenta con interfaces gráficas entre las cuales destacamos RStudio. [24]

En cuanto a las características del propio lenguaje, nos gustaría destacar las siguientes:

- Soporte a estructuras orientadas a la estadística como listas, matrices (Y arimétrica de matrices) o Data Frames, equivalentes a una tabla en una Base de Datos relacional y que veremos posteriormente, coordenadas geoespaciales, modelos de regresión y series temporales.
- Permite reescribir funciones en Fortran, C y C++ para casos computacionalmente intensivos.
- Permite la integración con diversas herramientas de reporting (Markdown con RMarkdown y Latex con Sweave o Knitr) y notebooks, principalmente Jupyter.

En cuanto a las ventajas y desventajas encontramos:

- Su fácil integración con Latex y varias de las herramientas de almacenamiento en la Nube permiten facilitar bastante las labores de escritura de la memoria de este proyecto así como la propia programación.
- Es un lenguaje pensado desde su especificación para labores como las que se llevarán a cabo en el trabajo, lo que garantiza su adecuación al proyecto.
- Facilidad para la visualización a través de paquetes como ggplot y broom.
- Sin embargo hay ocasiones en las que su uso no resulta tán fácil o por lo menos resulta más cómoda nuestra otra alternativa.

3.3.2. Python.

Python es uno de los lenguajes de programación más extendidos actualmente, considerado ampliamente como el tercero más popular detrás de Java y C aunque Python tiene a su favor una versatilidad espectacular, hasta el punto de que es el lenguaje de referencia en campos tales como Ciberseguridad, Animación y CAD, Scripting, Machine Learning e Inteligencia Artificial y también cuenta con una amplia presencia en el campo de la Minería de Datos. Python así mismo cuenta con una ya de por sí enorme librería estándar a la que si sumamos todos los paquetes y módulos accesibles a través de PIP (El instalador de paquetes de Python) se cuenta con un lenguaje treméndamente flexible y potente. [28]

Acerca de las características de Python nos gustaría resaltar las siguientes, para no extendernos excesivamente en este apartado:

- Python es un lenguaje multiparadigma capaz de funcionar como lenguaje compilado e interpretado y en programación procedural, orientada a objetos o funcional, entre otros, lo que explica en parte su gran acogida.
- La otra parte se debe a su amplia gama de librerías y paquetes, pensados para casi cualquier situación que deba afrontarse a la hora de programar.
- Python presenta un gran énfasis en la legibilidad del código, por lo que muchas de sus caracteristicas, como el uso extensivo de la tabulación están orientadas a esta finalidad.

Y a la hora de evaluar los puntos fuertes y débiles éstos son los más llamativos:

- Python también cuenta con una variedad de paquetes y librerías para interactuar con Latex y los diversos componentes de interés en una nube tan o incluso más extensa que la de R, destacamos entre ellos PythonTex, Matplotlib o ODBC.
- En relación a lo ya comentado, la capacidad multiparadigma de Python permite escribir programas con acceso a muchas más herramientas y técnicas de las que puede haber en R, cuya función primaria es la computación estadística.
- Python tiene un pequeño problema con la retrocompatibilidad pues las versiones actualmente soportadas del lenguaje (3.5 en adelante) no

pueden ejecutar código de las anteriores (Python 2 y en especial 2.7) debido a las nuevas características que aparecen en Python 3.

Puede así concluirse que R y Python presentan dos enfoques distintos para dar la misma funcionalidad: R está pensado y optimizado para este tipo de tareas mientras que Python apuesta por un conjunto de paquetes y una flexibilidad que le permiten solventar casi cualquier situación con una gran facilidad para el programador, de esta manera encontraremos situaciones donde pueda ser preferible el uso de R, como por ejemplo a la hora de integrar código en Latex, pues hemos comprobado que Pythontex da algunos problemas en la visualización de gráficas o compilando el documento; por la contra será preferible el uso de Python en situaciones como el manejo de conexiones con la Nube o en la creación de scripts.

Sin embargo, para elegir entre un lenguaje u otro no pueden ser considerados de forma aislada sino que deberán tenerse en cuenta las interfaces con las que deberán interactuar, las tareas que deberán llevar a cabo y en general su encaje con el resto de elementos de nuestro sistema.

3.4. Nube.

Antes de comenzar con el estudio de la compartición bajo demanda de recursos computacionales vía Internet resulta necesario definir algunos conceptos previamente, para poder entender la popularidad de la nube frente al modelo habitual de software en servidores propios, conocido como 'en premisas':

- Nube Privada: Son aquellas destinadas al uso exclusivo por parte de la empresa creadora, requiere de grandes medidas de seguridad y presenta unos costes muy elevados pero igualmente garantiza alta accesibilidad.
- Nube Pública: En las nubes públicas el servicio pertenece al proveedor pero el uso es abierto y no reside solo en la propia compañía, se caracterizan además por su infraestructura multiuso.
- Nube Híbrida: Aquellas que combinan soluciones públicas y privadas.
- SaaS: Software como servicio, se refiere a servicios ubicados en la nube que proporcionan acceso a aplicaciones software a consumidores a través de la web en un modelo de suscripción.
- IaaS: Infraestructura como servicio, proporciona acceso a recursos informáticos: Servidores, almacenamiento, direcciones IP y redes... Dentro de un entorno virtual.

- PaaS: Plataforma como servicio, un entorno de desarrollo en la nube; al igual que IaaS, PaaS incluye infraestructura pero también todo lo necesario para sustentar el ciclo de vida completo de una aplicación.

Actualmente existen 3 nubes consideradas como referencias: Google Cloud, Amazon Web Services, más conocida como AWS y Microsoft Azure, aunque hay otras como las de IBM, Oracle o AliBaba. Por cuestiones de familiaridad, extensión del trabajo y cuota de mercado nos centraremos en Azure y AWS. Ambas nubes cubren las tres categorías descritas anteriormente (Aunque sobre todo PaaS y IaaS) y entran en la categoría de nubes públicas, lo que nos servirá para más adelante establecer una comparativa entre ambas.

3.4.1. Azure.

Como ya no resulta necesario definir Azure (y AWS) gracias a las categorías anteriores, es posible describir diréctamente los servicios que proporciona Azure, de los cuales, son de interés para nuestro trabajo los siguientes:

3.4.1.1 HDInsight.

De forma concisa, HDInsight consiste en una distribución sobre la nube de Apache Hadoop que permite ejecutar sobre ella Spark, HBase, Storm o Kafka entre otros en base al Blob Storage de Azure, pudiendo manejar de serie varios lenguajes y pudiendo además visualizarse los datos mediante PowerBI o Grafana, Esto nos permite una gran flexibilidad a la hora de elegir el tipo de clúster que mejor se ajuste a nuestras necesidades. En concreto nos gustaría destacar dos, que son los que consideramos cumplen el criterio anterior: [14]

- ML Services, que a su vez es una adaptación de Spark, permite acceso bajo demanda y escalable para acceder a métodos distribuídos de analítica, principalmente en R, aunque tiene el relativo inconveniente de que no soporta Grafana (puede trabajar con PowerBI).
- Spark como standalone, como comentábamos antes nos permite un mayor grado de rendimiento que Hadoop y resulta posible integrarlo además con Python a través de Visual Studio Code y PySpark.

La documentación que aparece en la web de Azure es lo suficientemente extensa como para no recoger su funcionamiento ni su rango entero de precios,

sin embargo sí nos interesa exponer una parte para permitir una comparación, tanto por los clústers como por la gama de máquinas (Incluimos únicamente un elemento representativo de cada tipo de máquina):

Componente	Precio
Hadoop, Spark, Kafka, HBase...	Precio Clúster.
Machine Learning Services	Precio Clúster + €0,014/núcleo-hora
Enterprise Security Package	Precio Clúster + €0,009/núcleo-hora

Tabla 2: Precio de los frameworks en HDInsight

Instancia	VCPUs	RAM	S.O	Precio	Total
E4 v3	4	32 GB	0,258€/hora	€0,064/hora	€0,321/hora
D14 v2	16	112 GB	1,009€/hora	€0,253/hora	€1,261/hora
F16	16	32 GB	€0,767/hora	€0.257/hora	€1,023/hora
A8m v2	8	64 GB	€0,464/hora	€0,253/hora	€0,576/hora

Tabla 3: Precio de algunos clústers en HDInsight

3.4.1.2 Databricks.

Al contrario que HDInsight que tiene como base Hadoop, Databricks se construye diréctamente sobre Spark y es una plataforma analática que proporciona un entorno de trabajo interactivo entre ingenieros, analistas y científicos de datos, o lo que es lo mismo, aúna minería de datos, Big Data y visualización en la nube de manera que constituye una especie de todo-en-uno. Entre sus ventajas cabe destacar que ya viene diréctamente con soporte para el uso de múltiples lenguajes, tanto Python (mediante Koala) como R (Mediante SparkR) o Scala así como otras herramientas de Azure como por ejemplo Cognitive Search o CosmoDB.

En cuanto a sus precios Databricks ofrece tres cargas de trabajos a elegir según nuestras necesidades vayan por la ejecución de trabajos o sea necesaria también la visualización de datos. En las siguientes tablas podemos ver las distintas cargas de trabajo y algunas de las máquinas que aparecen en la documentación de Databricks: [14]

Componente	Precio Base	Precio Prémium
Análisis de datos	€0,34/DBU por hora.	€0,464/DBU por hora
Ingeniería de Datos	€0,13/DBU por hora	€0,253/DBU por hora
Ingeniería de Datos ligera	€0,06/DBU por hora	€0,186/DBU por hora

Tabla 4: Precio de las cargas de trabajo en Databricks

Instancia	VCPUs	RAM	DBUs	Precio
DS4 v2	8	28 GB	1,5	€1,155/hora
D32s v3	32	128 GB	6	€4,403/hora
L8s	8	64 GB	2	€1,556/hora
F16s v2	16	32 GB	4	€2,510/hora
H16	16	112 GB	4	€3,597/hora

Tabla 5: Precio de algunas máquinas en Databricks

3.4.1.3 Azure Cognitive Search, Data Explorer y Elasticsearch en Azure

Cognitive Search y Data Explorer son los dos servicios de Azure para analítica de datos, ligeramente diferentes en sus características y funcionalidad, en concreto Cognitive Search es la implementación de Azure de Lucene y puede integrar visualización a través de PowerBi. Data Explorer por la contra es un servicio original de Azure y está pensado para funcionar con un lenguaje de manipulación de datos, llamado Kusto Query Language y como su nombre indica está basado en SQL, Data Explorar para la visualización por la contra funciona con Grafana. [14]

En cuanto a los precios, está en nuestro interés considerar las opciones más baratas debido a que para este caso las prestaciones resultan de importancia secundaria al no llevarse a cabo cálculos tan intensivos, de esta manera las opciones a considerar son dos: [14]

- Cognitive Search tiene una opción gratis sin embargo su almacenamiento (50MB) no resulta adecuado para nuestros propósitos y deberemos optar por la opción estándar, con un almacenamiento de 25GB y un coste de €0,284/hora.
- Data Explorer cuenta con un coste base de €0.093/núcleo junto con un rango de máquinas similar al de los productos antes descritos, sin embargo tiene una opción especial para testeo de aplicaciones con 75GB de disco SSD cuyo coste es €0.161/hora.

Por último entre los servicios de Azure recogemos el conjunto de los distintos elementos del stack de Elasticsearch. Entre las ventajas de esta implementación está el soporte completo a información GeoSpacial en vez del parcial existente en Cognitive Search además de estar pensado para elaboración de reportes mediante Kibana. Elasticsearch además tiene tres paquetes para su gestión en Azure los cuales cuestan respectivamente 16, 19 y 22 dólares.

3.4.2. AWS.

Como es lógico, AWS presenta una gama de productos y servicios hasta cierto punto distinta a la de Azure, si bien a grandes rasgos cubre las mismas necesidades y funcionalidades y por tanto es posible implementar nuestro proyecto indistintamente en Azure o AWS, hablaremos aquí de los productos análogos a los descritos para Azure.

3.4.2.1 EMR.

Amazon EMR, siglas de Elastic Map Reduce y el homólogo de HDInsight, es la implementación cloud propietaria de Hadoop. Se ejecuta sobre la propia infraestructura de AWS: EC2/Elastic Compute Cloud para los clusters (Cada clúster es una colección de instancias de EC2) y opcionalmente S3/Simple Storage Service para el almacenamiento. En cuanto al funcionamiento y arquitectura de los clústers, éstos funcionan sobre HDFS o EMRFS (su sistema de archivos propietario e implementado sobre S3) es posible correr Hadoop de Map-Reduce o Spark y posteriormente montar HBase o Hive entre otros. [13]

EMR resulta gratis los primeros 12 meses; en la siguiente tabla recogemos parte de la gama de precios (En dólares, 1 Euro=1.1709 dólares) y máquinas, aunque no aparecen las características de las máquinas al no estar presentes en la documentación: [13]

Máquina	Tipo	Precio EC2	Precio EMR
m5.4xlarge	Uso general	0,768 USD por hora	0,192 USD por hora
c5.9xlarge	Optimización para informática	1,53 USD por hora	0,27 USD por hora
g4dn.4xlarge	Instancias con GPU	1,204 USD por hora	0,27 USD por hora
r5.8xlarge	Optimización para memoria	2,016 USD por hora	0,27 USD por hora
i3en.6xlarge	Optimización para almacenamiento	2,712 USD por hora	0,27 USD por hora

Tabla 6: Precio de varias máquinas en EMR

En cuanto a la comparativa con HDInsight, siendo ambos los ejemplos de "Big Data como servicio" estudiados y que será vista en detalle en el apartado siguiente, es posible ver de momento que:

- Azure presenta una mayor flexibilidad en su oferta de clusters que AWS pues Azure aparte de Hadoop y Spark permite usar Storm, Kafka, HBase y su cluster propietario pero que AWS cuenta con más variedad de máquinas virtuales que Azure.
- En cuanto a rendimiento no hemos visto ninguna diferencia apreciable pero si es cierto que Azure resulta ligeramente mejor a la hora de integrarse en Windows e interoperar con herramientas externas.

- En cuanto a precio resulta ligeramente mejor EMR tanto por su año gratis como por sus costes por hora.

3.4.2.2 Glue+Databricks y Glue+Elasticsearch.

Amazon Glue es un servicio ETL (Extract, Transform, Load) que posibilita cargar los datos, categorizarlos y moverlos a través de las herramientas pertinentes de análisis y almacenamiento. Entre sus puntos fuertes destacan que es serverless por lo que no requiere infraestructura y en su repositorio central (AWS Glue Data Catalog) se genera automáticamente el código necesario. Al final su función es la de preparar los datos para ser manipulados por otros servicios. [13]

Aquí es donde entra la implementación de AWS de Databricks, la cual no vamos a describir por ser similar a la de Azure. En los dos aspectos donde resultan diferentes es en la flexibilidad que le aporta Glue y el resto de herramientas de AWS (RedShift y SageMaker) a la implementación de AWS y en el precio, donde Azure similar a AWS pues AWS aporta 3 tarifas y dos cargas de trabajo de manera que los precios resultan similares, pero AWS no incluye a priori los precios de las instancias de las máquinas virtuales necesarias sino que vienen en la calculadora de precios, donde vemos que resultan similares a las ya descritas para EMR.

Elasticsearch para AWS se describe también aquí en esta sección pues emplea las mismas tecnologías y a grandes rasgos es similar a su funcionamiento en Azure, permitiendo también la integración con el resto de elementos del Stack de Elasticsearch, en cuanto a precios son los mismos que los descritos para Azure.

4. Implementación.

Una vez se han descrito tanto los datos que almacenará nuestro proyecto así como las distintas opciones existentes para trabajar con ellos puede hablarse ya de las elecciones en cada uno de los aspectos descritos en el apartado anterior; entramos pues en la segunda parte del trabajo, donde aparece recogido y explicado el proceso de diseño e implementación del entorno que da título a este trabajo. Dicha implementación puede dividirse a su vez según el almacenamiento de los datos, situados en la nube en alguno de los productos ya citados, el proceso de Extracción del conocimiento, donde se llevará a cabo la Minería de Datos y por último la integración de los distintos componentes junto a la visualización de los resultados obtenidos, considerando además el empleo de Dashboards para automatizar y agilizar dicha visualización en caso de resultar posible.

Cabe señalar aquí que se ha dado un cierto órden en la elaboración de los apartados para poder avanzar en paralelo en la elaboración del trabajo así como en los aspectos técnicos del mismo, de esta manera los datos fueron elegidos en primer lugar una vez se definió el concepto del trabajo para posteriormente centrarnos en la elección del lenguaje a emplear en la parte de minería de datos, posteriormente se tomó la decisión acerca de la nube idónea y finalmente los productos como servicio concretos para implementar e integrar los distintos componentes de nuestro desarrollo.

4.1. Elección de Arquitectura y Diseño.

En la elaboración de un flujo de datos adecuado en consideración a las herramientas elegidas para el almacenamiento, extracción de conocimiento y visualización deben tenerse primero en cuenta las distintas maneras de estructurar y conectar los distintos componentes de nuestro sistema, para ello expondremos aquí las arquitecturas que consideramos como viables, en función del tipo de servicio que quiera emplearse para después de explicar los pros y contras de cada una, compararlas como ya venimos haciendo anteriormente y pasar a la elección de lenguaje de programación y nube.

4.1.1. Propuesta 1: HDInsight.

Para la primera propuesta, si bien podría pensarse que HDInsight y EMR llevan a cabo la misma función y al construirse con los mismos elementos pueden agruparse bajo el mismo esquema, debido a las diferencias en la

manera que ambos productos estructuran su arquitectura y oferta de clústers esto no resulta posible.

Concretamente las diferencias son la manera de comunicar Hadoop con los lenguajes de programación/consulta, HDInsight oculta el funcionamiento de su arquitectura interna y permite la conexión directa con R y con Python (a través de PySpark), mientras que en el caso de EMR es necesario emplear Hive o Script Runner como intermediarios, con HDFS o con EMRFS de sistema de archivos y Map-Reduce o Spark como motor de computación sobre los cuales se agregan el resto de componentes, lo cual tiene como consecuencia que el código que proporcionamos a EMR debe ir a través de Notebooks. Por todo esto podemos concluir que EMR no resulta idóneo para nuestros propósitos y queda descartado como propuesta viable.

Comenzamos pues con la descripción de la arquitectura que emplea HDInsight, a lo que primero resulta necesario necesario clarificar el tipo de máquina y clúster que serán empleados y para ello debemos tener en cuenta las siguientes consideraciones:

- Anteriormente no se han considerado todas las máquinas que ofrece Azure para HDInsight por cuestiones de extensión, de manera que la máquina óptima que no venga recogida en las tablas anteriores.
- La característica de interés principal de cara al hosteo de nuestros datos es la memoria, seguida del precio y por último el número de CPU's. Esto es así debido a que trabajar con estos datos puede resultar computacionalmente intensivo (Como se verá posteriormente en el apartado de Minería de Datos) y además está en nuestro interés reducir costes debido a la naturaleza de suscripción con la que opera la nube.

De esta manera vemos lo siguiente:

- La máquina más adecuada para nuestros propósitos es la E16 v3, debido a su elevada memoria (128 GB), número de CPUs (16) y precio manejable, en caso de resultar demasiado grande como para nuestros propósitos, podríamos usar en cambio la A8m v2, que es a grandes rasgos la mitad de sus prestaciones. [14]
- En cuanto al clúster, por simplicidad de manejo y puesta en marcha junto a las posibilidades que ofrece y pese a ser un poco más caro que Spark, encontramos que la opción más adecuada es el clúster propietario de Azure, MLServices, esto es así debido a que su opción de enlazar diréctamente mediante R scripts en un ordenador local para

ejecutarlos en el propio clúster simplifica significativamente nuestro sistema.

De esta manera ya podemos describir en un diagrama la organización que siguen nuestros componentes, que viene representada en la siguiente figura:

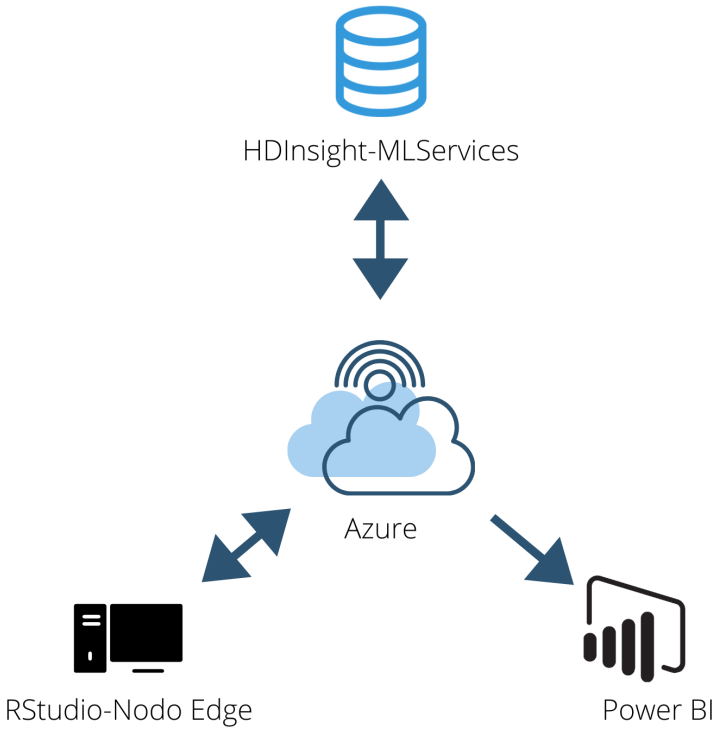

Figura 1: Esquema de la primera arquitectura: HDInsight

4.1.2. Propuesta 2: Databricks.

En el caso de Databricks, si que nos resulta indiferente la nube que elijamos como infraestructura puesto que a efectos prácticos ambas nubes hostean el mismo producto y aunque la configuración será distinta pues AWS emplea herramientas auxiliares para construir una tubería de datos el uso general, las instancias elegidas de la máquina virtual y el esquema general serán similares. Visto lo cual elegiremos Azure para describir esta propuesta por cuestión de homogeneidad con los otros posibles diseños y la máquina. En cuanto a los criterios para elegir la máquina virtual son a grandes rasgos los mismos que en la sección anterior, con la diferencia de que en Databricks tenemos un límite de ”cores”, es decir, estamos limitados en el tamaño y prestaciones de los entornos que podemos montar, habida cuenta de esta situación nos decantamos por la máquina DS3v2, con 4 CPUs, 14 GB de RAM y 0,595€/hora + 0,75 como coste de uso de DBU. [14]

En cuanto a las diferencias con el esquema de HDInsight se puede pensar de forma reduccionista que se trata de una especie de todo-en-uno en contraposición al encaje de piezas que supone emplear Hadoop en la nube, no es así y existen diferencias que sin ser pros y contras justifican que hablemos de propuestas distintas, entre ellas podemos citar:

- En un primer lugar obviamente el coste superior, tanto en las máquinas virtuales como en el coste del servicio.
- En relación con el anterior la facilidad de uso e integración con otras herramientas, en HDInsight también resulta posible pero no están presentes las mismas facilidades.
- Esa facilidad de integración también hace que sea posible realizar nuestros Scripts de minería de datos en otros lenguajes aparte de R, concretamente Python, Java y Scala (Como veremos también para el apartado siguiente), a priori esto no debería de ser un problema pero puede que deba tenerse en cuenta en caso de que alguna herramienta sólamente funcione para un lenguaje concreto.

Es por esto que una vez dada esta comparativa deberemos concluir cual de las dos alternativas junto a la siguiente resulta más indicada pero como conclusiones del trabajo podemos indicar las siguientes:

- En relación con la facilidad de uso de Databricks vemos también una mayor rigidez que implica no poder configurar nuestro sistema de la misma manera que podemos hacer con HDInsight.

- Sin embargo como veremos posteriormente también puede combinarse con las herramientas o lenguajes que nos interesen para formar arquitecturas no limitadas excluisamente a Databricks, lo que en un sentido compensa esa rigidez inicial.

Una vez tratado lo anterior podemos describir aquí el esquema resultante, Databricks además en el caso de R y Python para Azure se integra de manera distinta a como puede ser en HDInsight igual que para AWS, por lo que tiene sentido también hablar de arquitecturas distintas en ambas nubes, sin embargo como para este caso estamos únicamente considerando el caso de Azure, no vemos necesario representar aquí el esquema resultante para AWS, de esta manera, en Azure la figura quedaría de la siguiente manera:

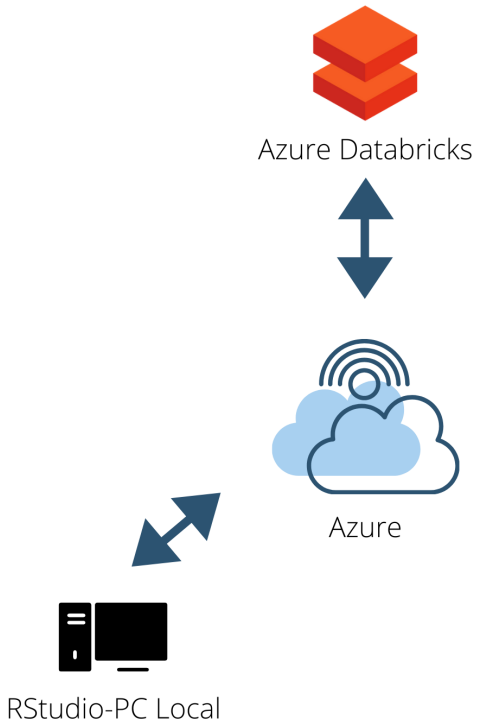

Figura 2: Esquema de la segunda arquitectura: Databricks

4.1.3. Propuesta 3: Cognitive Search/Databricks

Como el título de esta sección indica, primeramente esta propuesta consta de una arquitectura conjunta entre Cognitive Search y Databricks empleado como clúster de Spark y herramienta integradora, antes de entrar en detalles sin embargo, nos gustaría explicar por qué no se ha considerado el uso de Data Explorer así como de Elasticsearch, esto es debido a los siguientes factores:

- Elasticsearch presenta un coste demasiado elevado y además de no ajustarse a los propósitos del entorno que queremos crear, resulta complicado de implantar.
- En cuanto a Data Explorer vemos que Kusto Query Language no nos permite tampoco explorar los datos de la manera que buscamos en este trabajo al parecerse demasiado a SQL por lo que también lo descartamos.

De esta manera y teniendo en cuenta la documentación presente en Azure, donde se adjunta una arquitectura conjunta para Databricks y Cognitive Search y los datos para los que pensamos aplicar este esquema (Entrevistas en la NPR), resulta factible considerarlo para el segundo desarrollo en caso de permitirlo los límites de tiempo y dinero. Dicha arquitectura en realidad es simple y viene ya en Azure, pero igualmente la describimos a continuación: azure

- Primero de todo debe indicarse que esta propuesta deberían usarse los datos del conjunto 3, pues no son indicados geoespaciales sino aquellos sobre los que pueden hacerse análisis cognitivos.
- Se usaría Databricks para crear un clúster de Spark al que luego se conecta una instancia de Cognitive Services, opcionalmente a través de contenedores, es decir a través de un clúster de Kubernetes, la instalación del Spark Helm Chart y por último a través de Helm instalar Cognitive Services, en caso de conjuntos de datos extremadamente grandes, que no resulta nuestro caso y complicaría excesivamente el "dibujo", se puede proceder a instalarlo diréctamente en Databricks vía Maven.
- Además, para este desarrollo concreto no se puede usar R por lo que por defecto resultaría necesario el uso de Python al no contemplar este trabajo el uso de Java o Scala.

De esta manera el sistema puede resumirse de la siguiente manera:

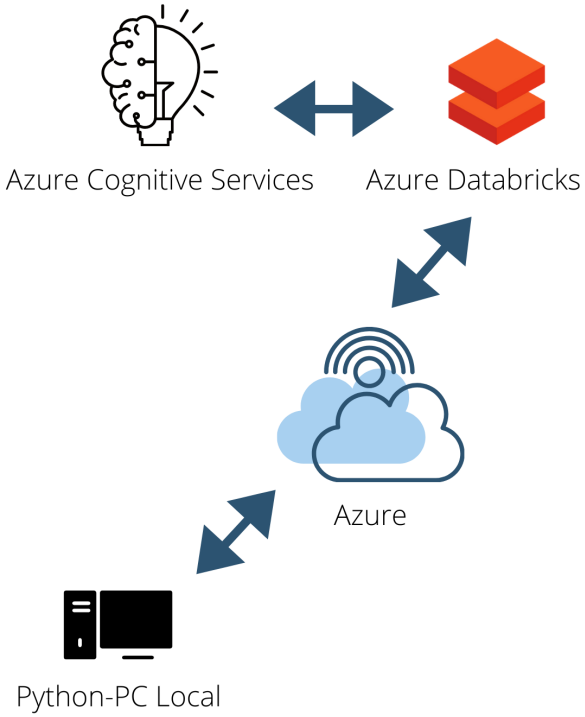

Figura 3: Esquema de la tercera arquitectura: Cognitive Services

4.1.4. Conclusiones del apartado y marco técnico

Elaborando sobre todo lo recogido a lo largo de todo este apartado y tratando también algunos de los puntos ya recogidos, podemos tomar una decisión sobre la arquitectura que resulta conveniente elegir y a la vez describir el marco técnico donde llevaremos a cabo la parte práctica del trabajo, es decir, el desglose de las herramientas, paquetes y entornos de programación que se han empleado en la parte práctica del proyecto.

Primeramente se ha tomado la decisión de seguir la segunda propuesta de implementación, es decir, emplear Azure y Databricks, esto es así debido a las siguientes razones:

- En primer lugar, era la única opción viable en términos de las restricciones de dinero bajo las que operamos de trabajar con Azure, no

requiere de dinero si se opera con la suscripción de prueba gratuita (aunque para usarla haya que actualizar la propia suscripción de Azure a pago-por-uso) y al estar integrado todo en la misma plataforma podemos a priori empezar a trabajar de manera casi inmediata.

- Si bien existe la opción de emplear también un clúster propietario proveniente de un trabajo anterior de mi tutor, se ve como prioridad trabajar con Azure pues al final resulta prioritario explorar desarrollos en las nubes comerciales y una vez realizado el trabajo en Databricks si el tiempo lo permite será posible tratar de replicar o mejorar los resultados obtenidos en el clúster disponible dentro del apartado de pruebas.

4.1.4.1 R, Notebooks y RStudio

El trabajo con R se hará principalmente desde los Notebooks de RMarkdown y RStudio, el entorno de desarrollo considerado como referencia; si bien existen otras alternativas, como por ejemplo la propia consola del lenguaje, RStudio integra de forma coherente los distintos elementos del ecosistema de R a través de su conexión directa con CRAN. Aunque de R ya se ha hecho una descripción donde era pertinente hacerlo, corresponde hacer un desglose de los paquetes empleados en los distintos scripts que componen la parte de programación del proyecto, dichos paquetes son bastantes y recogerlos a todos puede extenderse innecesariamente por lo que procuraremos recoger aquellos que sea imprescindible explicar.

- data.table contiene la función rbindlist que nos permite juntar una lista con los archivos .csv a una sola tabla que luego convertimos a dataframe.
- rgdal actúa como interfaz a las librerías y objetos geoespaciales de R, en concreto la usamos para leer el GeoJson.
- dplyr es un paquete de manipulación de datos pensado para trabajar con objetos de tipo dataframe, en este caso aporta funciones que nos permiten agrupar por varios parámetros y hacer tuberías en la manipulación de datos.
- ggplot2 es el paquete estándar de R para la visualización de datos y cuenta con multitud de opciones que le aportan gran versatilidad, sin embargo, es algo complicado de usar.
- broom permite preparar datos que ggplot no es capaz de manejar para su visualización a través de cambios de formato y estructura interna,

en concreto a través de las funcion tidy, aunque ggplot2 tiene una función parecida con fortify.

- dbscan es el paquete que contiene la implementación de diversos algoritmos de clusterización por densidad, descritos más abajo.
- Tanto maps como viridis son necesarias para que ggplot pueda visualizar mapas de calor, maps para funciones internas y viridis para el color y la escala.
- nominatim es un paquete que nos permite trabajar con la API de MapQuest, un servicio de geolocalización para realizar reverse geocoding, del cual hablaremos más adelante.

Un último apunte en este apartado es que aunque R es un lenguaje de programación interpretado, es decir, ejecuta línea a línea, como veremos a continuación para Databricks deja en parte de ser cierto.

4.1.4.2 Notebooks

Para el caso que de Databricks la única opción disponible para ejecutar código en R en la suscripción de prueba es la de emplear RMarkdown, RStudio solo resulta posible emplearlo en la suscripción premium, sin embargo sobre ambos corre la misma librería, SparkR, que resulta necesaria para leer los archivos y realizar las regresiones. Por lo demás son destacables las siguientes características y diferencias respecto de R como venía apareciendo hasta ahora.

- SparkR cuenta con tipos de datos distintos a los existentes en BaseR, en concreto la equivalencia general es la siguiente.
- Vemos también que SparkR genera conflictos por presentar funciones llamadas igual que las presentes en otros paquetes, por lo que se han tenido que renombrar varias funciones para evitar errores, por ejemplo group_by de Dplyr o Sample de BaseR.
- Si bien SparkR resulta necesario para leer los archivos, para poder aprovechar la flexibilidad que ofrecen los paquetes de R hemos convertido los Dataframes de Spark a los de R y a partir de ahí se ha trabajado toda transformación de los datos.
- En relación con los dos puntos anteriores, ha sido imposible trabajar con fechas y para los formatos y plots se han dado serias dificultades como veremos posteriormente, por lo que finalmente se han creado

varios workarounds para trabajar con Databricks, los cuales veremos posteriormente.

4.1.4.3 Visualización

Como se comentó anteriormente, uno de los inconvenientes de HDInsight es que no permite la asociación de sus clústers con Grafana en todos los casos y una de las excepciones es MLServices, por lo que la parte de Dashboards sería llevada a cabo con PowerBI, que es la herramienta de visualización propia del ecosistema de aplicaciones de Windows. Sin embargo al finalmente realizarse el trabajo con Databricks podemos usar la herramienta de visualización ya existente en Databricks, la cual nos permite elaborar distintos gráficos ya cargados previamentes y de los cuales aportamos un par de ejemplos, sin embargo, algo en lo que también haremos hincapié posteriormente es que aunque hayamos podido realizar visualizaciones desde Databricks no hemos conseguido replicar el funcionamiento de ggplot2 tal cual y como se presenta en RStudio.

4.2. Almacenamiento y Carga de datos.

Este apartado trata acerca del proceso seguido para la creación de los clústers y la subida de los datos en Azure, es decir la virtualización de los datos que se encuentran en nuestro ordenador físico. Por tanto a grandes rasgos va a tocar hablar del proceso de descarga y exploración preliminar de los datos, carga en azure, etc.

4.2.1. Exploración preliminar

A grandes rasgos aquí recogemos el proceso de descarga de los archivos. Sin embargo antes de comenzar con esta sección, es necesario indicar que hay varias versiones de los archivos y el órden con el que hemos trabajado con ellas para elaborar y probar los scripts de la parte práctica de este trabajo. Como se indicaba anteriormente en la primera descripción de los datos, primero trabajamos con los dos primeros días de la semana presente en Kaggle, una vez tuvimos el script preparado ha sido posible comprobarlo con los datos de Trentino y es aquí donde empiezan las complicaciones pues los conjuntos no son exactamente iguales:

- En primer lugar vemos que los archivos están en formato txt en vez de csv al abrirlos se parecen en bastante poco a la tabla que observábamos

en Kaggle, en primer lugar no tenemos nombres de columnas y la que da la fecha está en un formato distinto.

- Siguiendo lo anterior, la fecha del conjunto de Kaggle está en formato datetime y el número 'extraño' que se observa es también la fecha pero en una variente del formato Epoch, también conocido como tiempo POSIX o tiempo Unix, que es el tiempo transcurrido desde el 1 de enero de 1970, en este caso en milisegundos.
- Por último, los datos están agrupados por cada diez minutos en vez de por cada hora, puede parecer una diferencia trivial y al final lo acabará siendo, pero nos parecía relevante mencionarlo.

Antes de pasar a la siguiente sección vemos ilustrativo poner en una pequeña tabla un par de filas de los datos en bruto, para poder mostrar la estructura de los mismos, hay que señalar además que por temas de formato los decimales redondeados a 4 cifras e ISD representa el código del país hacia el que se produce la llamada:

ID	Tiempo	ISD	SMS-in	SMS-out	Call-in	Call-out	Internet
1	1383346800000	39	0.2724	0.1127	0.0035	0.0807	7.8378
1	1383412200000	39	0.7908	0.2452	0.8461	0.7683	14.3682
1	1383351000000	46					0.0261
1	1383351600000	0	0.0546				
1097	1383414000000	39	2.1589	1.2146	3.8515	3.3516	57.3198

Tabla 7: Datos TSV en bruto

4.2.2. Carga de datos en Azure

Una vez tenemos los datos descargados en nuestro ordenador, podemos comenzar a montar nuestro entorno, debido a asignaturas pasadas de la carrera, aunque tengo una cuenta como estudiante para Azure, dicha cuenta no cubre el acesso a los servicios que vamos a usar y resulta necesario empezar con el periodo de prueba de una cuenta nueva, lo cual sin embargo no afecta mucho a nuestro trabajo dado que el periodo de prueba de Databricks es de 14 días, sin embargo deberían ser suficientes habiendo dejado esta parte del trabajo para el final.

Una vez allí hemos creado un clúster mediante las siguientes especificaciones e instalado las librerías que eran necesarias para operar allí, que son las que aparecerán en las distintas celdas del notebook:

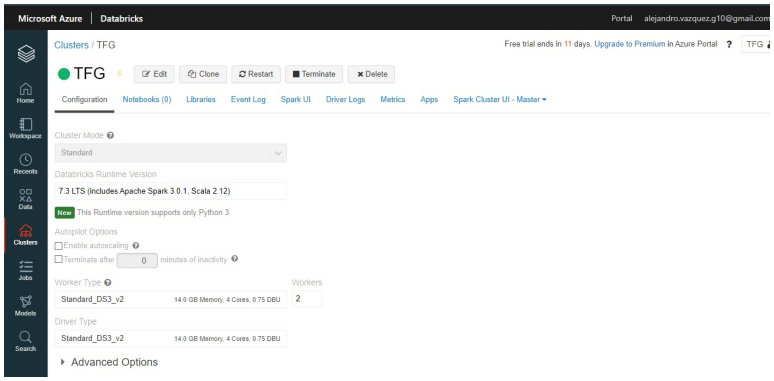

Figura 4: Especificaciones del clúster

En cuanto a la carga de datos propiamente dicha, una vez tenemos el clúster operando podemos subir los datos a través del propio Notebook en la ruta 'dbfs:/FileStore/shared$_u$*ploads*/*alejandro.vazquez.g*10@*gmail.com*/'.

Aquí nos hemos encontrado con el primer contratiempo de Spark/Databricks: No existe soporte a archivos de tipo TSV en las distintas librerías de Spark salvo en Scala, igualmente se ha visto que para cargar el GeoJson debe leerse como formato JSON plano de manera que si bien es posible acceder a sus contenidos, se pierde el esquema y resulta bastante laborioso transformarlo manualmente en un Dataframe. Por lo tanto que se decidió trabajar con la primera semana de noviembre en Milán en formato CSV y de la misma manera subir ya transformado a CSV la información geoespacial desde el ordenador local, facilitado por el hecho de que el propio asistente nos deja las líneas de código necesarias para leer el archivo (Aunque resultará necesario añadir el source y que el archivo emplea headers). Como puede

verse todo lo explicado en este apartado influenciará bastante el desarrollo y contenido del apartado posterior de la misma manera que las limitaciones presentes en Spark que mencionaremos a continuación son la causa principal de esta manera tan rebuscada de trabajar.

4.3. Extracción del Conocimiento.

En este apartado recogemos y detallamos la exploración realizada con los datos una vez ya se encuentran radicados en la nube y más concretamente en el sistema de archivos de Databricks; es aquí donde se concentra la mayoría de la programación realizada en este trabajo y donde describiremos la parte del trabajo concerniente a la minería de datos. Igualmente resulta intesante señalar que se han sucedido distintos enfoques a la hora de conceptualizar este apartado, por lo que no nos limitaremos a comentar los scripts en su versión definitiva sino a explicar todas las fases del proceso de desarrollo.

Otro de los puntos que resulta de interés comentar es que si bien generalmente el trabajo está escrito en órden cronológico, este punto constituye una excepción pues el desarrollo del apartado que nos ocupa ha sido anterior a la transición de los datos en la nube y la preparación del entorno a implantar. Esto es así debido a diversos factores, entre los cuales nos gustaría destacar los siguientes:

- El primero y principal es el ahorro de costes en vista del funcionamiento de la nube; está en nuestro interés tener el clúster desplegado el menor tiempo posible dado que Azure trabaja con un crédito inicial de 170€ y por tanto hemos visto preferible no comenzar con el proceso de virtualización de los datos y trabajar con los datos en premisas hasta tener esta parte perfilada y definida.
- En relación con el punto anterior, debe ser considerado también que la metodología de desarrollo que se ha seguido para trabajar en este punto se ha inspirado en el Prototipado y por tanto primeramente hemos trabajado con porciones pequeñas de los datos, las cuales perfectamente podían ser manejadas en premisas y es otro argumento a favor de por qué no comenzar por la nube.
- Al ser la parte del trabajo que consideramos como de mayor complejidad, también tiene sentido empezar a plantearlo y trabajar con el desde antes que los puntos considerados más fáciles, al final para subir los datos a la nube basta con seguir la guía que aparece en la documentación mientras que el desarrollo de los scripts de minería de datos, si bien es lícito reconocer que los notebooks existentes han servido en

ciertas partes como orientativos, especialmente a la hora de manipular y estructurar los datos en bruto, éstos se encuentran principalmente en Python cuando nosotros vamos a trabajar en R y además nuestro análisis no se centra en puntos concretos sino que trata de la comparativa entre los datos de Trento y Milán. [16] [1]

Una vez se ha introducido este apartado podemos pasar a describir los distintos enfoques que se han considerado para llevar a cabo la exploración de los datos.

4.3.1. Enfoque 1: Clusterización con Matriz de distancias geoespaciales

El primer concepto explorado para implementar un desarrollo fue emplear un algoritmo de clusterizado capaz de trabajar con coordenadas geoespaciales, como veremos adelante esto no ha sido posible debido a la propia estructura de los datos, sin embargo parecía un enfoque prometedor a la hora de visualizar la relación existente entre las distintas zonas de la ciudad pues como veremos en el siguiente gráfico perteneciente a la exploración preliminar de los datos, éstos parecen seguir una distribución de tipo 'campana' donde los valores más altos se concentran en el centro de la ciudad (Las celdas se numeran de abajo a arriba y de izquierda a derecha) tanto para el uso de internet como para las llamadas y mensajes.

Habida cuenta de esta situación se trató de probar si era posible mostrar esta situación de una forma menos cruda que un simple plot, por lo que se consideró hacer uso de la clusterización para mapear las distintas zonas según su valor de la variable que queremos destacar, para ello pensamos en usar los siguientes elementos.

4.3.1.1 K-Medias.

Como ya se mencionó anteriormente K-medias es un algoritmo de clusterización no supervisado diseñado para clasificar datos en un número K de particiones, de manera que los datos de cada clúster se parezcan entre ellos más que a los datos de clústers ajenos. En cuanto al funcionamiento interno del algoritmo, este no resulta muy complicado de explicar, la idea básica consiste en escoger un centroide inicial para cada clúster, asignar cada punto al centroide más cercano y posteriormente calcular de nuevo el centroide de cada cluster (según la función de distancia elegida) para actualizar el centroide al centro del clúster hasta que no se produzcan cambios en los puntos que conforman un clúster. [22]

R cuenta con una implementación de K-Medias así que no vemos necesario incluir el pseudocódigo del algoritmo porque además ya incluimos nuestro código más adelante.

4.3.1.2 Distancia Ortodrómica y del semiverseno/haversine.

Si bien por el tamaño de la ciudad de Milán puede considerarse que no se apreciaría una pérdida de precisión empleando la distancia Euclídea por ejemplo, en el caso de Trento si que se haría necesario calcular las distancias para la matriz del K-Medias con otra distancia, ahí entran las distancias que dan nombre a esta sección y cuya implementación en R hemos usado para calcular la distancia haversina entre los elementos de un dataframe, que es computacionalmente más eficiente que la alternativa de emplear función earth.dist del paquete fossil: [9]

Script 1: K-medias con distancias geodésicas

```
#Una vez tengamos las celdas agrupadas podemos clusterizar y conocemos el
    número de clusters
#Podemos hacer la clusterización (Por distancia geodésica)
#Función para la distancia geodésica
geo.dist = function(df) {
require(geosphere)
d <- function(i,z){              # z[1:2] contain long, lat
dist <- rep(0,nrow(z))
dist[i:nrow(z)] <- distHaversine(z[i:nrow(z),1:2],z[i,1:2])
return(dist)
}
dm <- do.call(cbind, lapply(1:nrow(df),d, df))
return(as.dist(dm))
}
print(prueba)
distancias <- geo.dist(prueba)
km<-bigkmeans(distancias,5, iter.max = 5,nstart = 1)
```

El problema más evidente que presentan estas fórmulas es que al ser mucho más precisas que empleando una distancia euclídea, también es mucho más intensiva computacionalmente de manera que se necesita una cantidad de memoria RAM prohibitiva para probar este código en un ordenador portatil o de sobremesa, en este caso se ha intentado recurrir a optimización del código como el uso de los paquetes especializados para computación de altas prestaciones de R (Big Matrix) o implementaciones de funciones en C++ para aumentar la eficiencia en memoria y se ha visto finalmente que no era necesario realizar un trabajo considerable en este aspecto dada la facilidad de acceso a recursos computacionales en cantidades considerables, como RAMs

de +64GB o +8CPUs que ofrece la nube.

Aun así el problema anterior es de carácter secundario respecto al problema principal en este enfoque y el siguiente. Sin embargo no aparece descrito aquí ya que fue a la hora de trabajar con los siguientes algoritmos donde nos dimos cuenta de lo que pasaba.

4.3.2. Enfoque 2: Clusterización basada en densidad

En vista de la dificultad de trabajar con el esquema anterior, se pensó posteriormente emplear la familia de los algoritmos de clusterización basada en densidad, es decir, OPTICS, DBSCAN, HDBSCAN, etc. El principal punto a favor de estos algoritmos es que no emplean una matriz de distancias en su implementación por índices al soportar aceleración por índices y son mucho más flexibles, eficientes en su rendimiento que K-Medias, además, pueden trabajar con clúster no separables linearmente, por lo que decidimos continuar usando DBSCAN. Los detalles teóricos del algoritmo vienen descritos en un paper considerado actualmente de referencia y que trataremos de resumir a continuación. [26]

Los dos parámetros principales del algoritmo son dos, el radio, comunmente representado con la letra épsilon (e) y los puntos mínimos o min-points, épsilon determina la longitud a partir de la cual si se incluyen los suficientes puntos que marca el otro parámetro consideraremos un área especialmente densa como un clúster, definido por esa densidad compartida y porque cada punto de un clúster es alcanzable con distancia e desde otro punto que a su vez cuenta con min-points a distancia e de él. En código de R quedaría de la siguiente manera: [25]

Script 2: Ejemplo de DBSCAN

```
prueba<-select(archivoGeoJson, long, lat, CellID,internet)
res <- dbscan(prueba, eps = 100, minPts = 5)
res
plot(prueba, col=res$cluster)
points(prueba[res$cluster==0,], pch = 3, col = "grey")
hullplot(prueba, res)
```

En la prueba de funcionamiento para el caso de Milán vemos que el resultado de la ejecución de DBSCAN ha servido efectivamente para detectar los valores atípicos de manera que DBSCAN selecciona en un solo cluster los datos que no se salen de la media para dejar fuera los valores atípicos, si bien como comprobaremos posteriormente, de manera que podemos descartar la necesidad de llevar a cabo este proceso pues estaríamos sesgando

de manera injustificada los datos, sin embargo en el caso de Trentino si se han producido que no se corresponden con lo esperado aunque puedan tener explicación, de esta manera adjuntamos aquí el resultado del algoritmo para los datos de Trentino:

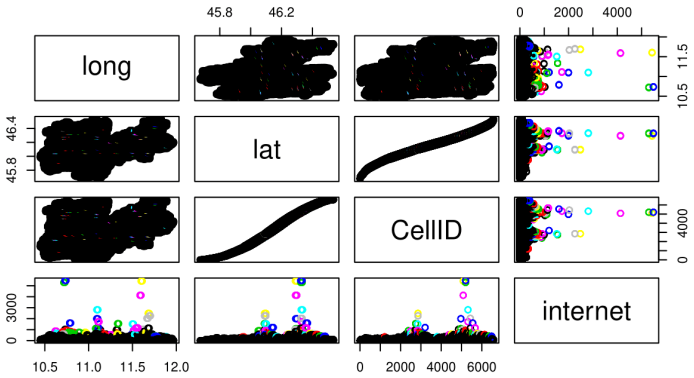

Figura 5: Resultado del DBSCAN en Trentino

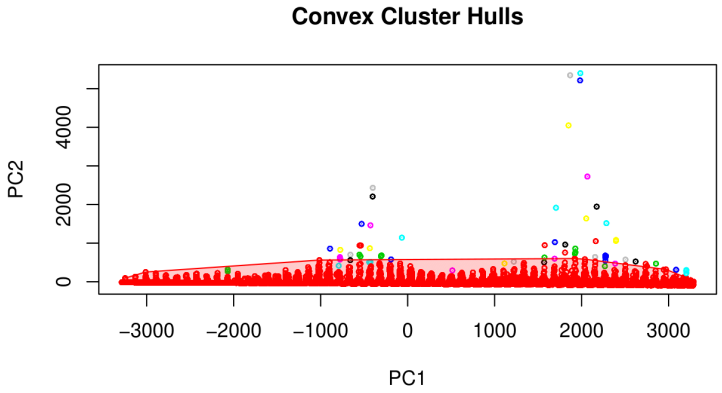

Figura 6: Resultado adicional con Hullplot

El problema que comparte este enfoque con el anterior y que impide obtener el resultado deseado es que si bien nuestros datos son estables, es decir, los clústers no varían de una ejecución del algoritmo a otra, al observar el resultado de DBSCAN y a partir de ahí las coordenadas de base, vemos que nuestros datos son de densidad uniforme, lo cual es debido a la estructura en cuadrícula (como indica el uso de la palabra grid) de los datos geoespaciales que proporciona nuestro conjunto de datos, de esta manera es imposible que

los datos estén más juntos unos a otros y si bien a priori DBSCAN resulta muy adecuado para casos como este la clusterización geoespacial no tiene sentido (Adjuntamos en la bibliografía un caso de uso de clusterización geoespacial para comparar). De esta manera si bien se ha acabado encontrando una situación que justificaba el uso del algoritmo como veremos más adelante, ha sido necesario también revisar lo que podía realizarse con nuestros datos y llegamos finalmente al estado actual para este apartado del trabajo.

4.3.3. Enfoque 3: Estado final

Finalmente tras comprobar la estructura interna de nuestro conjunto de datos y el funcionamiento de Spark en Azure se llega a la situación actual de este apartado, como el propio título indica se divide en dos partes complementarias, una centrada en mostrar la estructura y localización de los datos y la otra que trata de comprobar hasta que punto unas variables se encuentran relacionadas con otras. De los tres enfoques descritos este será naturalmente el más cargado de código pues aparte de incluir el código desarrollado, deberá de explicarse cuando sea preciso en tanto que la sintaxis de R puede resultar bastante enrevesada.

Este desarrollo que describiremos a continuación está implementado mediante scripts tanto en RStudio para un ordenador local como en Spark para realizar las transformaciones más intensivas computacionalmente (y con menos requerimientos de librerías especiales) y están dedicados a la visualización y exploración de los datos en Milán y Trentino, cabe destacar que para no duplicar el código generalmente solo mostraremos el código de Milán y en ocasiones este es el único existente, pues dado que R es un lenguaje interpretado, para obtener los datos de Trentino generalmente hemos reutilizado en la consola los programas en local para obtener los resultados al solo tener que modificar las rutas de los archivos.

4.3.3.1 Mapas coropléticos y Reverse Geocoding/Scripts de Visualización

Los mapas coropléticos, los cuales no deben confundirse con los mapas de calor, son uno de los medios más habituales para representar los distintos valores de una variable a lo largo de una región geográfica a través del sombreado del mapa mediante una gama de colores, de esta manerar resulta posible observar fácilmente la dispersión de la variable. Cabe señalar que para ello los datos deberán tener alguna relación con el espacio geográfico al que pertenecen y es posible que requieran de un tratamiento previo para poder conocer aquello que se desea resaltar o que los datos sean procesados

correctamente, en este caso mediante las herramientas de R. [29]

Esta parte de nuestro entorno se ha realizado mediante tres scripts en R en un ordenador local, uno auxiliar para poder presentar la información en formato legible para Spark, otro para visualizar los resultados obtenidos y uno último para realizar un ejemplo de geocodificación inversa. Junto a ellos está el notebook de Spark que lee los datos y realiza las transformaciones necesarias para poderlos visualizar,

Script 3: Carga de archivos a la nube

```
#Script que prepara archivos CSV para subirlos a Spark
library(rgdal)
library(broom)
library(data.table)
#Ruta, cambiar al directorio donde se guarden los archivos
setwd('C:\\Users\\GUILLERMO\\Desktop\\ProgramasVarios\\
    Python\\TFG\\archivos')
#Lectura de los archivos
#Los GeoJson
archivoGeoJson <- readOGR(dsn="milano-grid.geojson", layer=
    "milano-grid")
archivoGeoJson<-tidy(archivoGeoJson)
archivoGeoJson<-as.data.frame(archivoGeoJson)
names(archivoGeoJson) <- c("long","lat","order","hole","
    piece","group","CellID")
write.csv(archivoGeoJson,'csvmilano.csv')
archivoGeoJson <- readOGR(dsn="trentino-grid.geojson",
    layer="trentino-grid")
archivoGeoJson<-tidy(archivoGeoJson)
archivoGeoJson<-as.data.frame(archivoGeoJson)
names(archivoGeoJson) <- c("long","lat","order","hole","
    piece","group","CellID")
write.csv(archivoGeoJson,'csvtrentino.csv')
#El archivo tsv como csv
archivos <- list.files(pattern="sms-call-internet-tn
    -2013-11-0..txt")
auxiliarInternet <- list()
for (k in 1:length(archivos)){
  auxiliarInternet[[k]] <- setDT(read.table(archivos[k],sep
      = "\t"))
}
dataframeCDRs<-data.frame(rbindlist(auxiliarInternet, fill=
    TRUE, idcol=NULL),row.names=NULL)
#Como en el txt no vienen los nombres de las columnas, se los ponemos nosotros
#Según la documentación del conjunto
names(dataframeCDRs) <- c("CellID","datetimeAux","
    countrycode","smsin","smsout","callin","callout","
    internet")
#Es necesario pasar de Epoch a Central European Time y dropear los NAs para
```

```
    trabajar
dataframeCDRs$datetimeAux<-dataframeCDRs$datetimeAux/1000
dataframeCDRs$datetimeAux<-anytime(dataframeCDRs$
    datetimeAux, tz="CET")
dataframeCDRs[is.na(dataframeCDRs)] <- 0
write.csv(dataframeCDRs,'datostrentino.csv')
```

En este script el trabajo que se realiza es la lectura de los TSV de trentino y los archivos geoespaciales en formato GeoJSON. Para ello hacemos uso del paquete RGDAL, que nos permite leer los GeoJSON para después pasarlos a dataframe usando la función tidy del paquete Broom. En cuanto a la lectura de los archivos, éstos van primero como data.table para poder aplicar la función rbindlist y fusionar los distintos archivos de cada día en uno único. Una vez se encuentran los archivos como dataframes podemos guardarlos como CSVs y subirlos al sistema de Datos de Databricks.

Resulta importante señalar aquí cómo se han tratado las diferencias entre los datos existentes en Kaggle y los que encontramos tras descargarlos del repositorio que aparece como enlace en la documentación del conjunto de datos, que son las siguientes:

- A la hora de leer entre un archivo de tipo TSV y CSV no existe mucha diferencia, pero cambian las funciones de R y alguno de sus parámetros (En concreto el separador y el nombre (header) de las columnas, que no hay y debe añadirse, al menos trabajando en Spark.
- Los datos en el conjunto original se encuentran agrupados cada 10 minutos y en el de Kaggle por cada hora. No cambia el planteamiento y a la hora de hacer la agrupación salen valores casi idénticos parecidos pero no exáctamente iguales, lo cual es debido a que cambia un poco la operación y al aumento de la precisión en los datos, sin embargo se hizo necesaria una comprobación de resultados.

Para el caso del Script en Spark de lectura y transformación de los archivos de Milán, debido a su extensión vamos a dividirlo en varias partes que serán analizadas una a una:

Script 4: Lectura de los datos

```
#Formateo de datos inspirado en el Notebook Mobile Phone activity - exploratory
    analysis
#Librerías y directorio de datos
library(magrittr)
library(dplyr)      #Magrittr y Dplyr se usan para trabajar con tuberías
library(broom)      #Formateo de datos
library(splitstackshape)
```

```
library(data.table)
library(SparkR)
#Lectura de los archivos
#El GeoJson
coords1 <- as.data.frame(read.df("dbfs:/FileStore/shared_
    uploads/alejandro.vazquez.g10@gmail.com/csvmilano.csv",
    source = "csv", header="true"))
#Listamos, leemos los archivos (.. es cualquier carácter)
#Los pasamos a un dataframe Y los guardamos en auxiliarInternet,
#Se usa rbindlist porque es mucho más rápido y eficiente que docall
df1 <- as.data.frame(read.df("dbfs:/FileStore/shared_
    uploads/alejandro.vazquez.g10@gmail.com/sms_call_
    internet_mi_2013_11_01.csv", source = "csv", header="
    true"))
df2 <- as.data.frame(read.df("dbfs:/FileStore/shared_
    uploads/alejandro.vazquez.g10@gmail.com/sms_call_
    internet_mi_2013_11_02.csv", source = "csv", header="
    true"))
df3 <- as.data.frame(read.df("dbfs:/FileStore/shared_
    uploads/alejandro.vazquez.g10@gmail.com/sms_call_
    internet_mi_2013_11_07.csv", source = "csv", header="
    true"))
df4 <- as.data.frame(read.df("dbfs:/FileStore/shared_
    uploads/alejandro.vazquez.g10@gmail.com/sms_call_
    internet_mi_2013_11_06.csv", source = "csv", header="
    true"))
df5 <- as.data.frame(read.df("dbfs:/FileStore/shared_
    uploads/alejandro.vazquez.g10@gmail.com/sms_call_
    internet_mi_2013_11_03.csv", source = "csv", header="
    true"))
df6 <- as.data.frame(read.df("dbfs:/FileStore/shared_
    uploads/alejandro.vazquez.g10@gmail.com/sms_call_
    internet_mi_2013_11_04.csv", source = "csv", header="
    true"))
df7 <- as.data.frame(read.df("dbfs:/FileStore/shared_
    uploads/alejandro.vazquez.g10@gmail.com/sms_call_
    internet_mi_2013_11_05.csv", source = "csv", header="
    true"))
```

Esta parte del código no es especialmente complejo, ponemos las librerías necesarias y leemos los archivos del DBFS, es decir, del sistema de archivos de Databricks, para ello debemos especificar que leemos de un csv y que ya tiene header, aunque por si acaso volveremos a poner los nombres de las columnas pues Spark a veces ha cambiado algún nombre.

Script 5: Lectura de los datos en Databricks

```
auxiliarInternet <- list()
auxiliarInternet[[1]]<-df1
```

```
auxiliarInternet[[2]]<-df2
auxiliarInternet[[3]]<-df3
auxiliarInternet[[4]]<-df4
auxiliarInternet[[5]]<-df5
auxiliarInternet[[6]]<-df6
auxiliarInternet[[7]]<-df7
dataframeCDRs<-data.frame(rbindlist(auxiliarInternet, fill=
    TRUE, idcol=NULL),row.names=NULL)
#Como en el txt no venían los nombres de las columnas, se los ponemos nosotros
#Según la documentación del conjunto
names(dataframeCDRs) <- c("datetime","CellID","countrycode"
    ,"smsin","smsout","callin","callout","internet")
#Es necesario dropear los NAs para trabajar
dataframeCDRs[is.na(dataframeCDRs)] <- 0

#Juntamos las columnas de sms y llamadas
dataframeCDRs$smsin<-as.numeric(dataframeCDRs$smsin)
dataframeCDRs$smsout<-as.numeric(dataframeCDRs$smsout)
dataframeCDRs$callin<-as.numeric(dataframeCDRs$callin)
dataframeCDRs$callout<-as.numeric(dataframeCDRs$callout)
dataframeCDRs$internet<-as.numeric(dataframeCDRs$internet)
dataframeCDRs$sms<-dataframeCDRs$smsin+dataframeCDRs$smsout
dataframeCDRs$calls<-dataframeCDRs$callin+dataframeCDRs$
    callout
dataframeCDRs$smsin <- NULL
dataframeCDRs$smsout <- NULL
dataframeCDRs$callin <- NULL
dataframeCDRs$callout <- NULL
```

Aquí se procede a juntar los distintos archivos en un único dataframe mediante la función rbindlist, donde debemos convertir de vuelta a dataframe porque devuelve un datatable. Después eliminamos los NAs y casteamos las columnas a numeric para poder trabajar con ellas juntando sms y llamadas, pues si no R daría error al no poder trabajar con ese tipo de datos (Los provenientes de Spark), después borramos las columnas poniéndolas a NULL.

Script 6: Transformación de los datos en Databricks

```
summary1<-dataframeCDRs %>%
  dplyr::group_by(datetime,CellID) %>%
  summarise(internet=sum(internet),calls=sum(calls),sms=sum
      (sms))
dataframeCDRs<-as.data.frame(summary1)
summary2<-dataframeCDRs %>%
  dplyr::group_by(CellID) %>%
  summarise(internet=mean(internet),calls=mean(calls),sms=
      mean(sms))
dataframeCDRs<-as.data.frame(summary2)
```

```
dataframeCDRs<-dplyr::arrange(dataframeCDRs, "CellID")
#Por último como aquí Spark no consigue realizar la visualización
#Hacemos un left join y descargamos los datos para hacer la visualización en el
    ordenador local
archivoGeoJson <- merge(x=coords1,y=dataframeCDRs,by="
    CellID",all.x=TRUE)
archivoGeoJson[is.na(archivoGeoJson)] <- 0
display(archivoGeoJson)
```

Este código contiene primeramente una operación de agrupado en lo que se conoce como Tidyverse a través del paquete dplyr, entre cuyas características principales reside el uso de tuberías, resulta necesario emplear este paquete pues nuestra primera opción, la de usar aggregate, no era viable debido a los errores que daba tanto para trabajar por fechas como tal y de la función en sí. Concretamente lo que estamos haciendo es agrupar por fecha y CellID para despuer hacer de cada columna la suma total de esa hora como paso intermedio para después poder agrupar por medias en cada celda. Cabe destacar que normalmente no resulta necesario especificar el paquete de las funciones en R pero como comentábamos hay casos donde se producen conflictos y esta es la manera más sencilla, aunque no necesariamente elegante de evitar el error.

Habiendo realizado la parte del código anterior resulta posible hacer ahora un left join por CellID (Pues no queremos que desaparezcan datos de las Celdas ya que el plot quedaría a medias) del dataframe con la información geoespacial y el que contiene los datos ya trabajados. La exploración preliminar se ha llevado a cabo mediante simples plots que sin embargo permiten ir arrojando luz sobre la estructura e información de los datos, en este caso podemos ver que en las 3 variables los datos parecen seguir una distribución similar a la de campana o de Gauss, y como comentamos anteriormente que las celdas están numeradas de abajo arriba e izquierda a la derecha. Los valores centrales son los más altos, por tanto podemos intuir que la distribución de las variables es ascendiente y concéntrica, lo cual trataremos de confirmar con más exactitud con los mapas de calor.

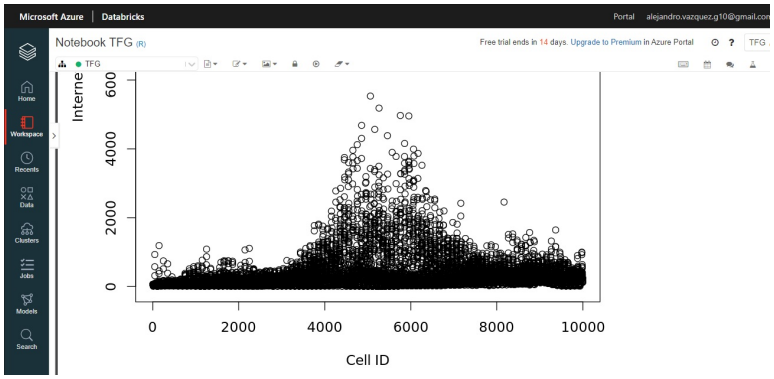

Figura 7: Primera visualización de los datos

Para ello primero será necesario preparar la información del GeoJson de manera que pueda entenderla R, por ensayo y error se ha visto que para ello es necesario cambiar la columna id a CellID (id es un nombre reservado y R levanta un error al intentar usarlo), ponerla como numeric o no la reconoce y una vez allí podemos hacer un left join para finalmente juntar todos los elementos en un solo dataframe y poder visualizar los resultados. Cabe realizar aquí un pequeño apunte y es que los archivos están numerados de manera distinta, es decir, el archivo geoespacial numera las celdas desde 0 y el archivo con la información de la actividad móvil desde 1, por lo tanto resulta también necesario en este caso poner los NAs a 0s, si no, daría error al realizar el plot.

Script 7: Visualización de los datos en BaseR

```
#Plots
library(ggplot2)
require(maps)
require(viridis)
library(dplyr)
#Ruta, cambiar al directorio donde se guarden los archivos
setwd('C:\\Users\\GUILLERMO\\Desktop\\ProgramasVarios\\
    Python\\TFG\\archivos')
#Lectura de los archivos

comprobacion<-as.data.frame(read.csv("exportComprobacion.
    csv"))
arrange(comprobacion,CellID)
archivoGeoJson<-as.data.frame(read.csv("export.csv"))
#Como en el txt no vienen los nombres de las columnas, se los ponemos nosotros
#Según la documentación del conjunto
#Nos sobra esta columna (Viene del Join de Spark y ahí no funcionaba esta línea)
archivoGeoJson$X_c0 <- NULL
#Guardamos los plots de Milán
```

```
pdf("plotInternetMilan.pdf", width = 8, height = 7)
ggplot(archivoGeoJson, aes(long, lat, group = group))+
  geom_polygon(aes(fill = internet), color = "white")+
   scale_fill_continuous(low='white', 'high'='red',limits=c
       (0,5000))
theme_classic()
dev.off()
pdf("plotLlamadasMilan.pdf", width = 8, height = 7)
ggplot(archivoGeoJson, aes(long, lat, group = group))+
  geom_polygon(aes(fill = calls), color = "white")+
   scale_fill_continuous(low='white', 'high'='red',limits=c
       (0,400))
theme_classic()
dev.off()
pdf("plotSMSMilan.pdf", width = 8, height = 7)
ggplot(archivoGeoJson, aes(long, lat, group = group))+
  geom_polygon(aes(fill = sms), color = "white")+
   scale_fill_continuous(low='white', 'high'='red',limits=c
       (0,600))
theme_classic()
dev.off()
#Guardamos los plots de Trentino
archivoGeoJson<-as.data.frame(read.csv("exportTrentino.csv"
    ))
archivoGeoJson$X_c0 <- NULL
pdf("plotInternetTrentino.pdf", width = 8, height = 7)
ggplot(archivoGeoJson, aes(long, lat, group = group))+
  geom_polygon(aes(fill = internet), color = "white")+
   scale_fill_continuous(low='white', 'high'='red',limits=c
       (0,200))
theme_classic()
dev.off()
pdf("plotLlamadasTrentino.pdf", width = 8, height = 7)
ggplot(archivoGeoJson, aes(long, lat, group = group))+
  geom_polygon(aes(fill = calls), color = "white")+
   scale_fill_continuous(low='white', 'high'='red',limits=c
       (0,100))
theme_classic()
dev.off()
pdf("plotSMSTrentino.pdf", width = 8, height = 7)
ggplot(archivoGeoJson, aes(long, lat, group = group))+
  geom_polygon(aes(fill = sms), color = "white")+
   scale_fill_continuous(low='white', 'high'='red',limits=c
       (0,100))
theme_classic()
dev.off()
```

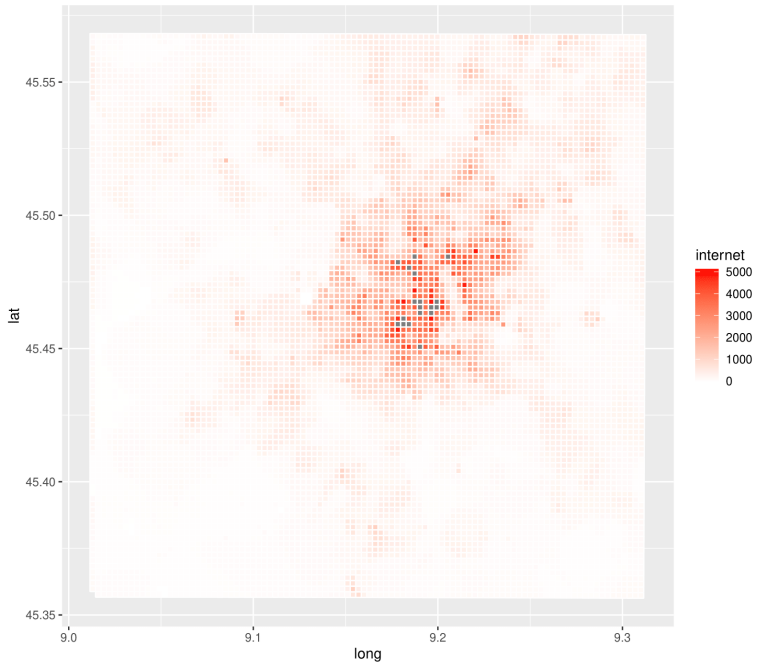

Figura 8: Mapa de Calor del Internet en Milán

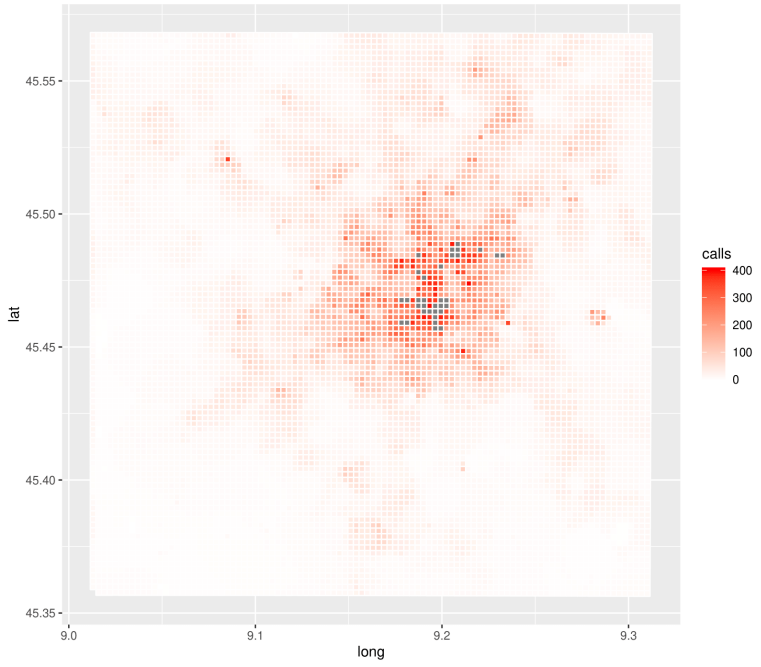

Figura 9: Mapa de Calor de las llamadas en Milán

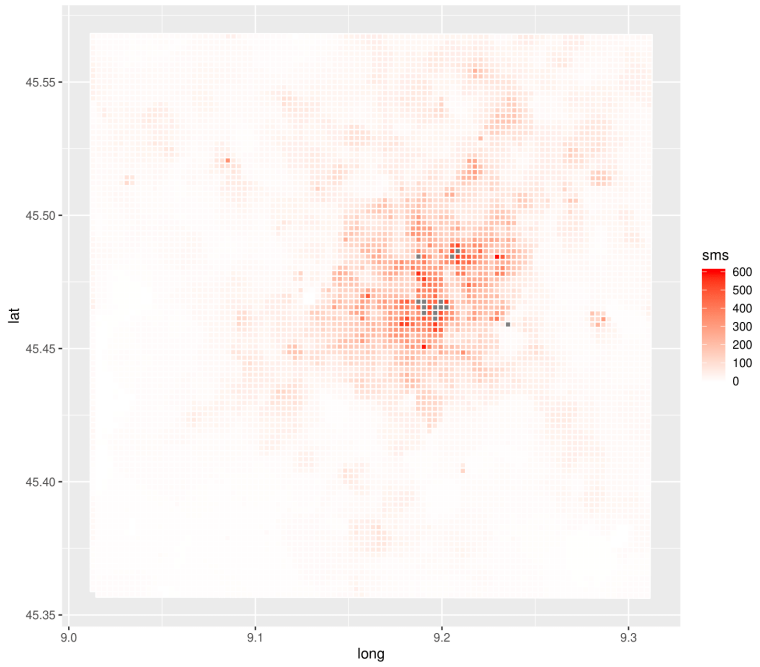

Figura 10: Mapa de Calor de los SMS en Milán

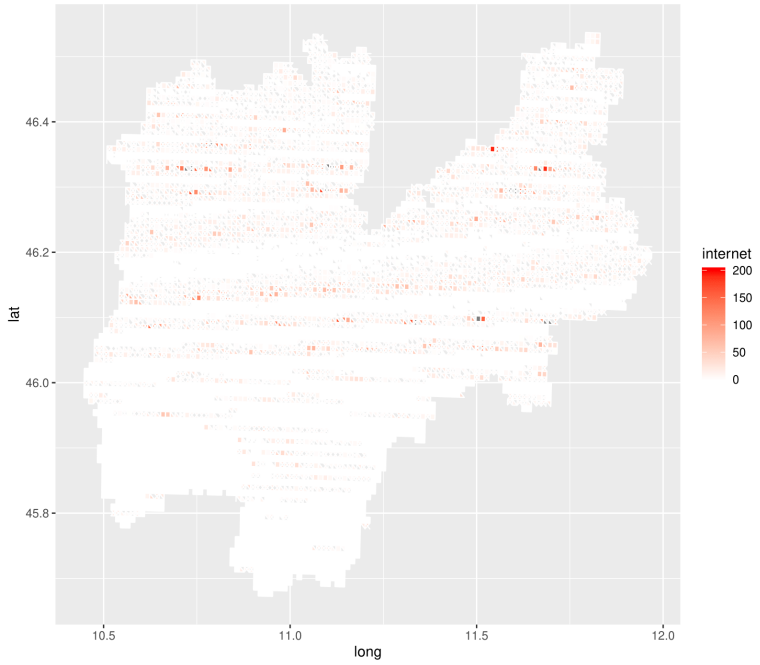

Figura 11: Mapa de Calor del Internet en Trentino

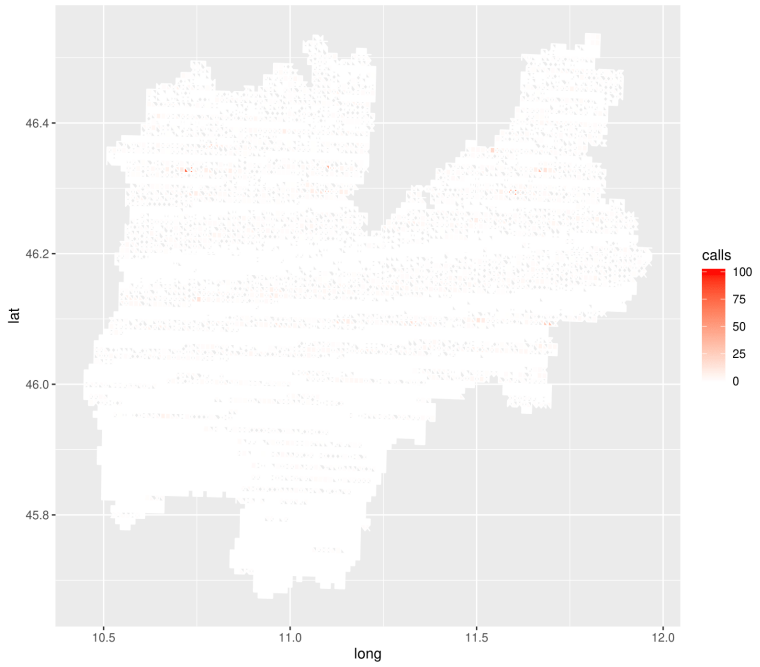

Figura 12: Mapa de Calor de las llamadas en Trentino

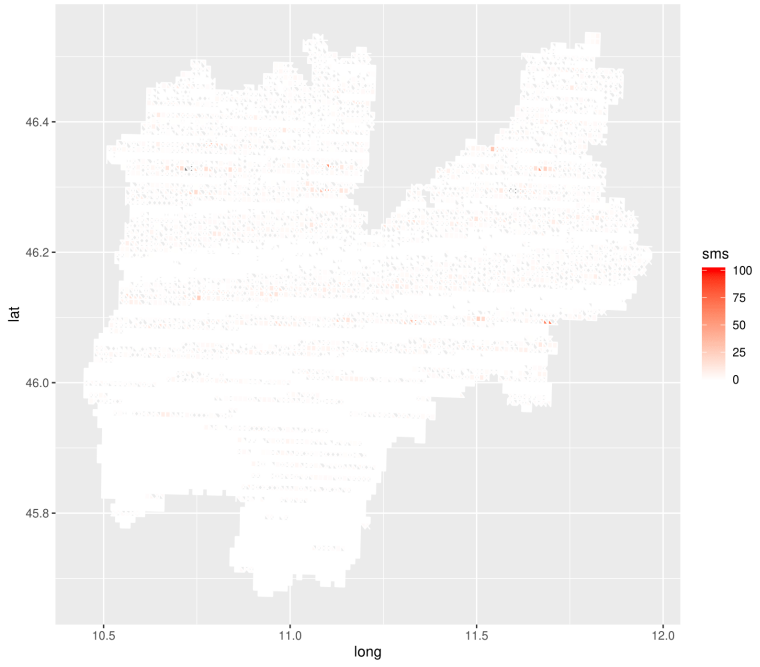

Figura 13: Mapa de Calor de los SMS en Trentino

Lo que vemos aquí es la lectura de los archivos en formato CSV procendentes de Spark (Los que llevan el nombre export) y su posterior visualización. Primeramente es importante comentar que si se ha optado por esta forma de trabajar es porque el mismo plot que aquí produce el resultado en Spark devuelve una traza distinta y el resultado que aparece por pantaña difícilmente puede ser considerado como satisfactorio, de manera que se ha optado por trabajar con el dataframe resultante de las celdas del Notebook de Spark, los cuales hemos comprobado que eran correctos cotejando los resultados de los dos primeros días en local y en Spark. Un apunte necesario es que para mostrar correctamente la escala debe ajustarse a la variable que se desea visualizar, tanto para el caso de Milán como el de Trentino. Es importante notar también que los guardamos en formato PDF para conservar la calidad de las visualizaciones, dev.off() se usa para cerrar y guardar una visualización y preparar la siguiente

Como podemos observar a continuación el código de cada plot es virtualmente idéntico y rellena y muestra cada uno de los polígonos geométricos que componen nuestro mapa, en este caso celdas. Como detalle destacamos que si bien la forma general de los plots es la misma, también la escala de cada plot debe ser ajustada 'manualmente' pues la actividad en cada una de las tres variables es disinta. En cuanto al resultado de los plots podemos comprobar efectivamente que el valor de cada variable observada aumenta conforme se acerca al centro urbano de Milán desde los suburbios, suponemos que esto es así porque es donde tiene lugar la mayor parte de la actividad económica y donde se concentra buena parte de la población durante su jornada laboral y tiempo de ocio. Esto último podría comprobarse haciendo una serie de plots hora a hora, pero en aras de una fácil lectura del trabajo, recogemos solamente los plots de las medias de todos los días.

En cuanto a los datos del Trentino puede intuirse en los plots preliminares que los datos van a presentar una estructura mucho más dispersa y con mucha menos magnitud en los datos, lo cual se confirma posteriormente viendo que la mayoría del mapa es blanco y las celdas con color se encuentran bastante dispersas en el mapa, sin embargo, los resultados no son exáctamente los esperados, esto es así porque adelantando resultados posteriores en las coordenadas más cercanas a Trento vemos que la actividad es francamente escasa y que los dos picos observados de actividad se encuentran en coordenadas bastante alejadas de cualquier núcleo poblacional. Además, se han revisado concienzudamente los archivos para descartar cualquier tipo de error por nuestra parte (los datos fuera de la norma se encuentran originalmente en dichos archivos). Es aquí donde observamos mediante geolocalización inversa que los valores más altos están ubicado razonablemente cerca de una de las zonas más populares de senderismo en el Trentino, por lo que podría explicarse dicha actividad en base a la consulta de rutas, tracks

y GPS. Existe una última opción que es la de suponer que ambos picos en la actividad se deben a erratas en la recolección de los datos o a actividades fuera de la norma. De esta manera podrán considerarse dichos valores como datos anómalos, por lo que tendrá sentido haber ejecutado el algoritmo DBSCAN en los datos de Trentino y haber obtenido los resultados ya mostrados anteriormente.

Teniendo en cuenta lo anterior, en cuanto a las conclusiones de este apartado podemos nombrar las siguientes:

- Se han encontrado diferencias apreciables tanto en magnitud como en la distribución de la actividad móvil, en Milán esta se encuentra agrupada en el centro de la ciudad mientras que en Trentino está distribuida a lo largo de todo el territorio con picos irregulares pero por lo general se aprecia mucha menos actividad.
- En la propia estructura interna de los datos, desde el comienzo de la numeración a la estructura, formato y presentación de los datos también se han encontrado situaciones inesperadas y que han hecho necesaria una profunda revisión de los datos y código en más de una ocasión.
- La anomalía en los datos de Trentino puede explicarse en base a los propios datos obtenidos, es decir, aquellos que consideramos como valores anómalos, pues ya se encuentran presentes los valores en los archivos que leemos (y por tanto no es problema de nuestro código) además de que que en la fecha en la que tuvo lugar el estudio no se ha encontrado ningún acontecimiento que pueda explicar con solidez dichos valores.

4.3.3.2 Geocodificación Inversa

La geocodificación inversa es un proceso que permite conocer a partir de una ubicación en un mapa, generalmente por coordenadas y más concretamente por latitud y longitud, una dirección legible; generalmente esto se encuentra implementado en servicios a los que nos conectamos mediante peticiones una vez suscritos (ArcGis, Google o en nuestro caso, Nominatim a través de MapQuest). El código no resulta de mucha complejidad pero nos permite comprobar donde se encuentran las coordenadas de ciertos resultados. Damos aquí un ejemplo de su uso. [27]

Script 8: Geocodificación inversa

```
archivoGeoJson<-as.data.frame(read.csv("export.csv"))
#Ejemplo
```

```
#Nos sobra esta columna (Viene del Join de Spark y ahí no funcionaba esta línea)
archivoGeoJson$X_c0 <- NULL
long<-archivoGeoJson[which.max(archivoGeoJson$internet),2]
lat<-archivoGeoJson[which.max(archivoGeoJson$internet),3]
internet<-archivoGeoJson[which.max(archivoGeoJson$internet)
    ,8]
print(long)
print(lat)
print(internet)
emb_coded_coords <- reverse_geocode_coords(lat, long,key='
    sYiO2A8cc6pAhsdykyERWWjzkc0NsGUu')

emb_coded_coords %>% select(display_name)
```

El código en si no tiene mucha complicación, buscamos a través de las coordenadas, en este caso aquellas cuyo valor de internet es el más alto registrado y a través de la clave que se nos da al darnos de alta en el servicio podemos sacar sus coordenadas. Concretamente vemos que las coordenadas están en pleno centro de Milán lo cual coincide con los resultados obtenidos anteriormente, sin embargo el problema radica cuando pasamos a los datos de Trentino, pues como comentábamos antes las coordenadas con los valores más elevados no se encuentran cerca de ningún núcleo poblacional.

4.3.3.3 Regresión lineal

La regresión es una herramienta matemática que permite predecir el valor de una variable según una o más variables, la regresión suele ser habitualmente lineal y en relación a una sola variable, pero existen modelos alternativos, como por ejemplo la regresión robusta, los cuales no exploraremos en este trabajo y regresiones multivariable también, las cuales en cambio son bastante fáciles de hacer en R.

En el caso de la regresión lineal adelantamos que debido a las anomalías de los datos en Trentino así como de la evidente dispersión de los datos que presenta dicho conjunto de datos, como ya encontramos anteriormente, será difícil encontrar correlaciones ahí y por tanto solamente realizaremos este script para el caso de Milán. El código comienza de manera similar al script anterior por lo que no lo incluiremos. Sin embargo hay una salvedad que debe ser mencionada y es que no sustituiremos los NAs hasta después de realizar la visualización del modelo ya que se ha visto que contando los NAs como ceros, la función aumenta considerablemente su tiempo de ejecución. El código como tal es el que sigue y funciona prácticamente igual en Databricks que en R normal:

Script 9: Visualización de las regresiones

```
#Para este script no resulta necesario realizar trasformaciones en el dataframe a
    priori
#Pair Scatterplot
pairs(dataframeCDRs)
#Scatterplot de todas las variables con su modelo, comprobamos una a una
#Primero hacemos un plot de la regresión para después construir el modelo y
#Contrastar con la predicción (Entrenamiento con train y test)
lmPlot(dataframeCDRs$callin, dataframeCDRs$callout) #Funciona
    bien
lmPlot(dataframeCDRs$smsin, dataframeCDRs$smsout) #También se
    observa correlación
lmPlot(dataframeCDRs$callin, dataframeCDRs$smsout) #Poca
    correlación
lmPlot(dataframeCDRs$callin, dataframeCDRs$smsin)
lmPlot(dataframeCDRs$callin, dataframeCDRs$internet) #Poca
    correlación
lmPlot(dataframeCDRs$smsout, dataframeCDRs$internet) #
    Poquísima
    correlación
```

Observamos aquí el resultado de nuestras comprobaciones (Mostramos primero el resultado obtenido en Databricks y luego los plots replicados porque no es posible descargarlos):

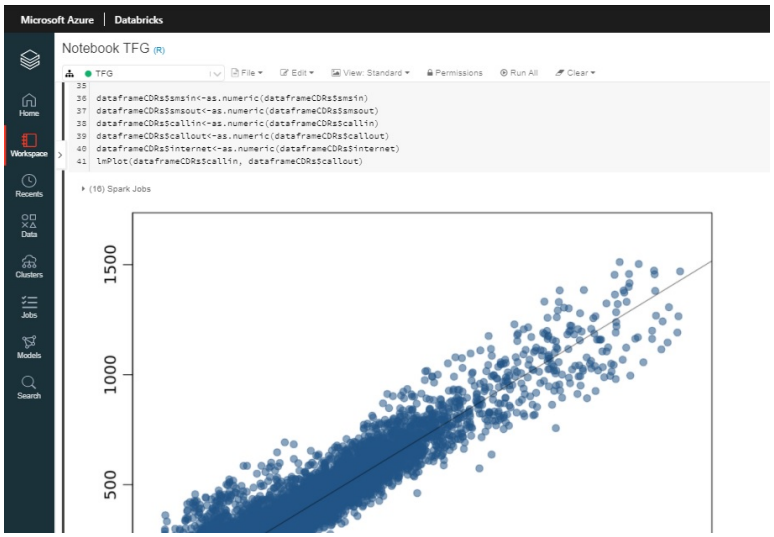

Figura 14: Resultado de la regresión con lmplot

Pero para empezar a sacar conclusiones sobre este apartado necesitaremos primero ver el resultado de trabajar con modelos en vez de con visualizaciones y para ello deberemos samplear los datos y entrenar un modelo.

Script 10: Ejemplo manual de una regresión en Databricks

```
indice <- base::sample(1:nrow(dataframeCDRs), 0.4*nrow(
    dataframeCDRs))
entrenamiento <- dataframeCDRs[indice, ]
test  <- dataframeCDRs[-indice, ]
linearMod <- glm(callout ~ callin , data=entrenamiento,
    family = "gaussian")
pred <- predict(linearMod, test)
summary(linearMod)
precision <- data.frame(cbind(datosVerdaderos=test$callin,
    predicciones=pred))
display(precision)
```

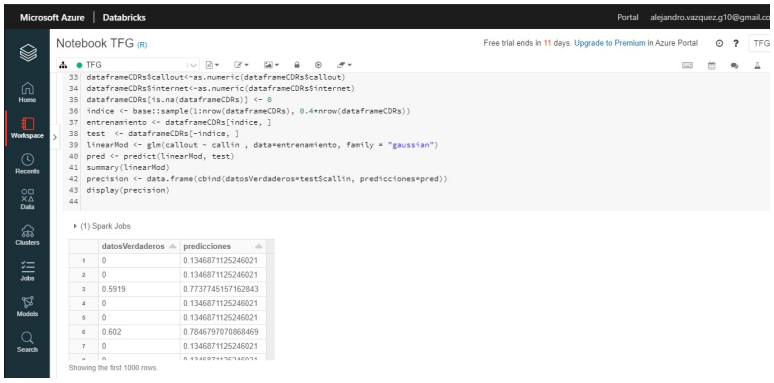

Figura 15: Resultado de la regresión

De esta manera lo que hacemos es dividir nuestros datos en dos conjuntos, el conjunto de entrenamiento y el conjunto de test, esto es así porque en Machine Learning suelen dividirse los conjuntos de datos en tres subconjuntos: training, test y validation, aunque nosotros únicamente necesitaremos los dos primeros conjuntos, de esta manera podremos crear nuestro modelo y realizar un tratamiento de regresión como suele ser habitual, es decir: entrenarlo mediante los datos acordes para poder realizar predicciones sobre los datos de test y compararlos con el resultado obtenido mediante los datos de entrenamiento.En cuanto a las conclusiones de analizar los datos mediante una regresión, vemos principalmente que las correlaciones fuertes princpalmente son entre las llamadas entrantes y salientes así como entre mensajes en un menor grado, con internet y regresiones multivariable apenas se han observado correlaciones.

4.4. Dashboards e Integración.

Para la última sección de este apartado hay relativamente poco que comentar en comparación con los anteriores, pues para explicar los resultados obtenidos era preciso mostrarlos en su totalidad y por tanto condición sine qua non mostrar los plots que nos permiten mostrar la estructura de los datos, sin embargo comentaremos aquí la parte de las dificultades técnicas y el proceso que emplea Databricks para la creación de Dashboards así como las relaciones existentes entre las distintas partes de Databricks y la manera en la que han afectado a nuestro trabajo.

4.4.0.1 Dashboards

Los dashboards en Databricks se realizan a partir de los Notebooks, de manera que es necesario primeramente crear uno y una vez tenemos los resultados creados podemos hacer click en view dashboard y las celdas se añaden automáticamente, de esta manera se ha creado un ejemplo de Dashboard que exponemos a continuación con los plots obtenidos finalmente en Databricks y un ejemplo de gráfico en SparkSQL siguiendo el ejemplo de la documentación. [15]

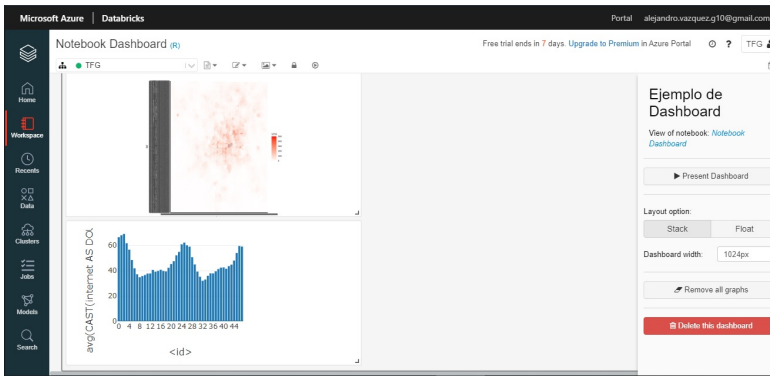

Figura 16: Ejemplo rudimentario de un informe

Un último apunte en este apartado es que los dashboards en Databricks cubren también la parte de reporting al poderse programar de forma periódica como un trabajo a ejecutar en el clúster, si bien presenta algunas particularidades como el hecho de que a cada ejecución se crea un nuevo clúster y que insertar texto debe hacerse mediante HTML lo cual es bastante farragoso. Sin embargo con suficiente trabajo puede hacerse un dashboard mucho más

completo que el esqueleto que presentamos nosotros, que, sin embargo, sirve para demostrar el funcionamiento de la creación de informes en Databricks.

4.4.0.2 Integración

A la hora de hablar de los distintos componentes que conforman nuestro sistema, debido a la elección tomada entre las posibles alternativas resulta más adecuado hablar de la relación existente entre las distintas partes de Databricks, lo cual no es demérito de Databricks pues es precisamente uno de sus puntos fuertes y nos ahorra redactar buena parte de lo que vendría recogido en esta sección de haber empleado HDInsight.

Cualquiera que sea el caso, al hacer login en Databricks vemos los distintos apartados del área de trabajo, los distintos menús para administrar datos, trabajos clústers y modelos, es notable también destacar las acciones más frecuentes (Creación de un notebook, tablas, clúster, trabajos/jobs, experimentos, importación de librerías y por último un enlace de acceso a su documentación). De esto podemos sintetizar que todos los elementos que pueden necesitarse para montar un desarrollo se encuentran fácilmente al alcance de cualquier programador con cierta experiencia, lo cual dice mucho y muy bien de la integración que han logrado entre Spark, Spark SQL, los distintos lenguajes de programación que admite, en más de alguna manera en algunos casos, la capacidad de enlazar Dashboards al resultado de los notebooks o la de realizar tuberías de Machine Learning con MLFlow y dar monitoreo y soporte a los modelos obtenidos.

Estos son los puntos fuertes de la plataforma, en cuanto a los negativos principalmente se dan dos: Unos es que algunas de las mejores características y por tanto el pleno potencial de Databricks se encuentran en la suscripción premium, lo cual ha afectado en ocasiones a nuestro trabajo pues habiendo la intención de realizar los scripts de R se han tenido que realizar en notebooks, que si bien ha automatizado considerablemente la parte de conexión al clúster nos lleva al segundo problema de Databricks, interconectar tantas herramientas y darles soporte supone que en algunos casos van a existir problemas o no se va a poder soportar la funcionalidad completa de la herramienta original. Hemos hablado de esto en detalle cuando comentábamos las dificultades de programar R en Spark sin embargo también se han notado problemas a la hora de crear el clúster, instalar ciertas librerías y en la permanencia de resultados, a veces el notebook daba error en código que previamente había dejado de funcionar corréctamente y era necesario volver a adjuntar el notebook al TFG o reiniciar el clúster.

Por tanto podemos concluir que si bien presenta ciertos problemas, especial-

mente si se elige trabajar con R en vez de con Python o Scala, Databricks es una plataforma muy dada para trabajar con el tipo de entornos que se plantean en este TFG pues nos proporciona una arquitectura muy completa con sus distintos componentes ya desplegados y una documentación razonablemente decente, si bien requiere de tiempo y dinero para llegar a sacarle todas sus capacidades.

5. Pruebas y Casos de Uso.

Al trabajar con arquitecturas y tecnologías propietarias junto a las ya indicadas restricciones de tiempo y dinero la capacidad de replicar y comprobar resultados es limitada. Sin embargo se ha tratado de rehacer el trabajo con Spark a modo de comprobante de su capacidad de ser llevado a otros entornos en un clúster con el sistema operativo Debian 9 que mi tutor del TFG venía empleando de trabajos anteriores. El resultado es que siguiendo la documentación de SparkR puede replicarse parte del código del desarrollo en Databricks, aunque hay aspectos que han condicionado bastante la capacidad de reproducir este resultado, por lo que para obtener el resultado deseado se ha tenido que emplear SparklyR, dando así a una especie de comparativa entre ambos:

- A la hora de instalar Spark nos hemos visto condicionados por el entorno pues la versión de Spark soportada (3.0.1) difería con la de SparkR(2.4.6) y en la ejecución del código aparecían bastantes warnings relacionados con este aspecto. De la misma manera, hemos tenido que emplear versiones antiguas de algunos paquetes porque la versión de BaseR existente en los paquetes de Debian no admitía las versiones actualizadas y se ha debido recurrir al archivo de CRAN.
- Seguidamente en base a lo anterior no se ha podido trasladar el Dataframe que lee Spark al de R propiamente dicho de la misma manera que no hemos podido usar tampoco correctamente Sparklyr (suponemos que por las limitaciones de memoria del Driver de Spark o la falta de un área de Swap en el clúster), lo cual ha complicado significativamente nuestras tareas.
- En el caso de SparkR hemos tenido que adaptar buena parte del código a la manera de trabajar que viene en la documentación de SparkR con la cual al no poder trabajar con las estructuras de R ha sido dificil adaptar el código. Entre las dificultades que han tenido lugar son destacables los casteos automáticos que realiza, el tratamiento de errores bastante deficiente que muestra trazas considerablemente largas de

errores de Scala o Java que se arreglan con una sola línea de R o problemas con la propia interfaz de Spark, pues por ejemplo para hacer una regresión lineal gaussiana hemos tenido que emplear únicamente 4096 filas (máximo permitido según la documentación de Spark) y en Databricks se ha podido hacer sin tantas dificultades.

- Con SparklyR en cambio si se ha podido obtener un resultado más satisfactorio, concretamente realizando la regresión con los conjuntos en un tamaño correcto aunque también se han tenido problemas en los mismos aspectos que SparkR (lectura de archivos, memoria del driver) y aunque la documentación es ligeramente más orientativa y fácil de seguir que la de SparkR, ha costado bastante trabajo hacer que funcione una porción de código conceptualmente bastante poco complejo y que se adaptadaba de un entorno sobre el papel casi idéntico.

Finalmente dejamos aquí los resultados obtenidos entre SparklyR y SparkR para la realización de un ejemplo de regresión. No se ha realizado el ejemplo de los mapas de calor en parte por no repetir el trabajo hecho y especialmente porque dado que no se ha podido hacer con SparkR no queremos llegar a la falsa conclusión de que uno es mejor que otro:

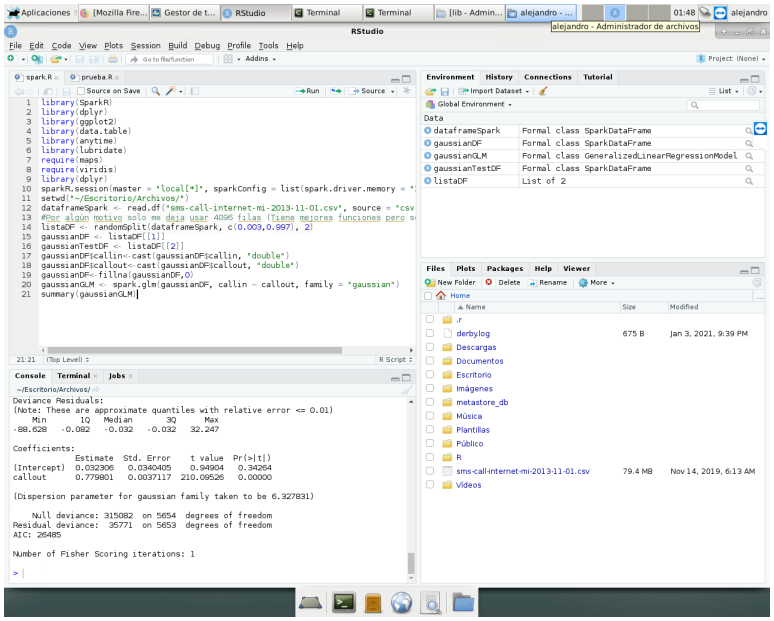

Figura 17: Resultados con SparkR

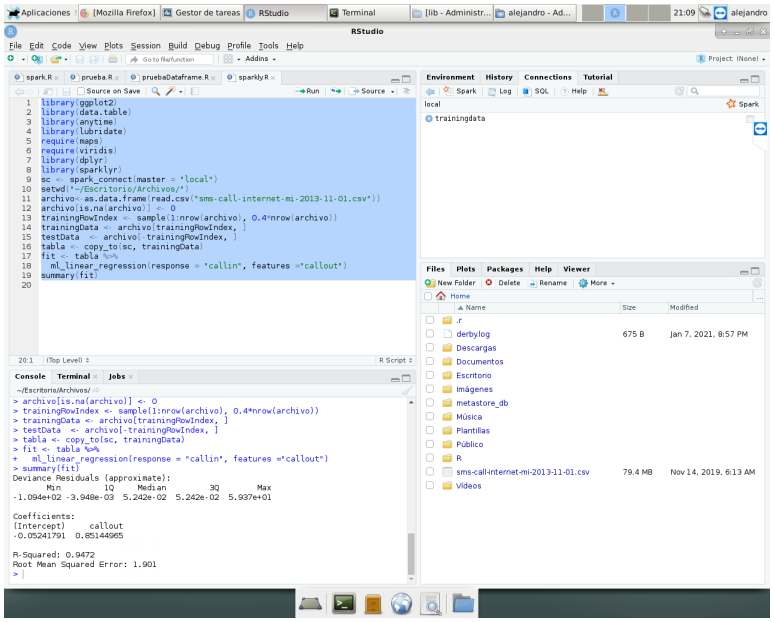

Figura 18: Resultados con Sparkly

6. Presupuesto.

Dado que la parte práctica del trabajo se ha realizado con elementos de Plataforma e Infraestructura como servicio hemos visto adecuado presentar aquí un pequeño desglose de los costes de este proyecto y del tiempo dedicado a las tareas realizadas en términos de horas trabajadas como en el coste actual debido a los pagos a a Azure.

Teniendo en cuenta que el trabajo previo de estudio duró aproximadamente dos meses y que el trabajo de programación se realizó concurrentemente con el de elaboración de este documento y dos asignaturas de la carrera, hemos realizado una estimación en la que a tiempo completo las tareas de programación han durado en torno a un mes. Teniendo en cuenta los precios de Databricks, es decir 0,06€ la hora y 0,595€ el coste de la máquina trabajando alrededor de 6 horas al día es posile suponer un coste de 50 a 70€ dependiendo del número de días útiles siendo mi propia estimación de 51'09€. De de haber trabajado con la suscripción premium como se tenía pensado, al ser 0,186€ el coste de la hora el coste se dispara a torno a 100-125€ para trabajar con RStudio. Si a esto se suman costes adicionales como el salario del trabajador, posibles planes de soporte, el precio de los cursos en la academia de Databricks ($75 por sesin, 1500$ por curso completo.) Se comprende que trabajar con estas tecnologías implica un coste considerable para lo cual una empresa que no esté afianzada en su negocio quizá no pueda diréctamente ni asumir o no le resulte rentable en comparación a tecnologías similares.

Sin embargo esta estimación no ha resultado ser correcta en vista de la factura que resultó del uso de Azure, sin contar los 170 euros de crédito de prueba y que hemos excedido, el coste ha sido de 237,70€ brutos, que en total tras el descuento queda en 81,92€, lo cual supone un coste prohibitivo para cualquier desarrollo que pretenda explorar las posibilidades de la nube. En cuanto al clúster propiedad del tutor de este trabajo, no se ha incurrido en ningún coste por emplearlo debido a que ya estaba pagado de antemano, pero a juzgar por nuestra experiencia se trata de una instalación que resulta más compleja de emplear que Databricks por lo que el coste de los desarrolladores podría resultar bastante más elevado aunque sea difícil de estimar.

7. Conclusiones.

A la hora de hablar de conclusiones hay que diferenciar entre aquellas que pertenecen a las aseveraciones sobre la propia manera de articular el trabajo y la arquitectura del mismo (Nube, productos como servicio), aquellas que nacen de la comprensión y exploración de los datos y su estructura (Como las diferencias entre el conjunto de datos de Milán y el del Trentino) y aquellas que nacen de la propia experiencia técnica en la implementación del proyecto (Consideraciones sobre el manejo de R o los clústers en la nube). Principalmente serán las primeras las que más importancia merezcan en este apartado al tratarse de un apartado general, pero recogeremos también aquellas que traten de las particularidades del trabajo y sean relevantes en la medida que influyan en el funcionamiento completo del sistema:

- Resulta obvio que el funcionamiento de los productos, infraestructuras y plataformas como servicio está pensado para generar beneficios a la empresa proveedora del servicio, sin embargo lo que nos ha resultado llamativo es que la curva de precios aumenta de manera pronunciadísima a medida que se requieren mayores prestaciones o rendimiento en el servicio, por ejemplo un servicio de geolocalización puede tener 1000 peticiones gratis al día o 15000 al mes y en cuanto sobrepasas de esa métrica son 100€ al mes en adelante. Resulta igual para la nube en el rango de máquinas o el de los planes de soporte; por tanto concluimos que la virtualización si bien cuenta con los beneficios obvios desde el punto de vista de la escalabilidad, soporte y disponibilidad, puede incurrir en un coste muy elevado para una empresa de tipo start-up si no se planea adecuadamente, como es la perspectiva de este trabajo o incluso resultar contraproducente, por tanto nuestra conclusión y recomendación es acompañar cualquier estudio acerca de una posible virtualización con un análisis de riesgos y estudio de viabilidad.
- En cuanto a la comparativa entre nubes así como en la de lenguajes de programación no se ha podido determinar un claro "vencedor". En el caso de Azure y AWS la diferencia es dificil de encontrar hasta el punto de que la decisión de usar una u otra se ha tomado en base a consideraciones secundarias pero la versatilidad de Azure se equilibra con la competitividad en los precios de AWS y a la hora de comparar R con Python las diferencias son también secundarias, Python presenta un mayor control de memoria pero R hace ciertas aspectos de los scripts bastante menos complejos para el programador; la única comparativa donde hemos visto que una propuesta sea mejor a otra ha sido en el Big Data donde Spark y sus derivados aún con su problemática resultaban francamente superiores o por lo menos más accesibles a las alternativas

existentes para los propósitos de este trabajo.

- Se ha podido comprobar que efectivamente existe un nivel significativamente menor de actividad móvil en el Trentino que en Milán, así como que dicha actividad se encuentra de forma dispersa en el Trentino y concentrada de forma concéntrica en Milán, lo cual corresponde con el estatus de Milán como ciudad global y con el carácter de provincia rural del Trentino, se ha observado también cierta correlación entre el uso de llamadas y mensajes tanto entrantes como salientes en Milán, sin embargo el uso de internet está bastante poco relacionado con el resto de variables, lo que creemos que puede explicarse en base al carácter autosuficiente del uso de internet, nacido de la extensión masiva de los smartphones y de las aplicaciones de mensajería instantánea.

- A nivel de memoria se ha visto que R carece de un recolector de basura en tiempo de ejecución por lo que un uso prolongado del mismo con la cantidad de datos que se puede manejar en un entorno de estas características requiere de optimización del código para evitar que R acabe acumulando en su sesión y entorno variables que ya no utiliza y puedan comprometer el funcionamiento normal del sistema. No es que R no sea eficiente en términos de memoria o rendimiento, sobre todo con el uso de paquetes especializados, sin embargo es necesaria la limpieza periódica de memoria. Además, en cuanto a la intercompatbilidad entre funciones y librerías, SparkR y Spark han dado problemas y no ha sido posible acceder a la funcionalidad completa por lo que podemos añadir también que resulta muy recomendable conocer bien el entorno y la versión de los paquetes con los que se está trabajando.

Referencias

[1] Barlacchi, G., De Nadai, M., Larcher, R., Casella, A., Chitic, C., Torrisi, G., ... Lepri, B. (2015). A multi-source dataset of urban life in the city of Milan and the Province of Trentino. Scientific data, 2(1), 1-15.

[2] Prasad Majumder, B., Li, S., Ni, J., McAuley, J. (2020). Interview: A Large-Scale Open-Source Corpus of Media Dialog. arXiv, arXiv-2004.

[3] Enlace a los datos de Madrid en la página web de Murray Cox: http://insideairbnb.com/get-the-data.html

[4] Enlace a la competición que da origen a los datos de italia: http://www.telecomitalia.com/tit/en/bigdatachallenge.html

[5] Documentación de Hadoop: https://hadoop.apache.org/docs/current/

[6] Introducción a Hadoop: https://intellipaat.com/blog/tutorial/hadoop-tutorial/introduction-hadoop/

[7] Documentación de Spark: https://spark.apache.org/docs/3.0.1/

[8] Tutorial/Introducción a Spark: https://www.infoq.com/articles/apache-spark-introduction/

[9] Discusión acerca de como clusterizar datos geoespaciales: https://stackoverflow.com/questions/21095643/approaches-for-spatial-geodesic-latitude-longitude-clustering-in-r-with-geodesic

[10] Discusión en Stack Overflow que optimiza los métodos anteriores: https://stackoverflow.com/questions/34991978/dbscan-for-clustering-data-by-location-and-density

[11] Discusión en Stack Overflow sobre los datos etiquetados: https://stackoverflow.com/questions/19170603/what-is-the-difference-between-labeled-and-unlabeled-data

[12] Documentación de Elasticsearch: https://www.elastic.co/guide/index.html

[13] Documentación de AWS: https://docs.aws.amazon.com/index.html

[14] Documentación de Azure: https://docs.microsoft.com/en-us/azure/?product=featured

[15] Documentación de Databricks:https://docs.databricks.com/index.html

[16] Tutorial de Azure acerca de cómo usar R y el empleo de una regresión: https://docs.microsoft.com/en-us/azure/hdinsight/r-server/ml-services-tutorial-spark-compute

[17] Tutorial de R acerca de modelos de regresión: http://r-statistics.co/Linear-Regression.html

[18] Notebook de Kaggle donde se plantea una regresión sobre el dataframe: https://www.kaggle.com/altanai/linearregession-sms-callmobile-data

[19] Notebook de uno de los autores usado como referencia para plantear nuestro análisis: https://www.kaggle.com/gbarlacchi/mobile-phone-activity-exploratory-analysis

[20] Artículo de Oracle de donde tomamos la definición de Big Data: https://www.oracle.com/big-data/what-is-big-data.html

[21] Artículo de la Enciclopedia Británica en el Data Mining: https://www.britannica.com/technology/data-mining

[22] Artículo sobre el K-medias: https://towardsdatascience.com/k-means-a-complete-introduction-1702af9cd8c

[23] Caso de uso DBSCAN: https://bitsandbricks.github.io/post/dbscan-machine-learning-para-detectar-centros-de-actividad-urbana/

[24] Artículo en CRAN sobre R: https://www.r-project.org/about.html

[25] Documentación del paquete dbscan de R: https://www.rdocumentation.org/packages/dbscan/versions/1.1-5

[26] Artículo original del DBSCAN: https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf

[27] Artículo y documentación sobre reverse geocoding en R: https://www.r-bloggers.com/2015/07/introducing-the-nominatim-geocoding-package/

[28] Guía sobre Python: https://wiki.python.org/moin/BeginnersGuide/Overview

[29] Guía sobre machine learning con datos geoespaciales: https://heartbeat.fritz.ai/working-with-geospatial-data-in-machine-learning-ad4097c7228d

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá