uc3m | Universidad Carlos III de Madrid

Ingeniería Informática Superior
2016-2017

Trabajo Fin de Carrera

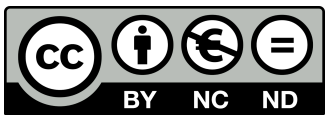
"Sistema Recomendador de Taxis para *Big Data*"

Jorge Barata González

Tutor/es

Pablo Basanta Val

Leganés, 20 de Septiembre de 2017



Título: SISTEMA RECOMENDADOR DE TAXIS PARA BIG DATA
Autor: Jorge Barata González
Director: Pablo Basanta Val

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Gracias a Pablo, que aún sin conocerme no dudó en ofrecerse a ayudarme a sacar adelante mi última asignatura pendiente: el Proyecto de Final de Carrera.

Resumen

Este documento propone un sistema para maximizar los ingresos de los taxistas, recomendando las zonas de la ciudad con mayor frecuencia de viajes y más cercanas a la posición del taxista en ese momento.

Analizando 10M de trazas de viajes realizados por los Taxis Amarillos de Nueva York con un cluster Spark, encontramos correlaciones entre los beneficios y el tiempo, distancia y número de pasajeros. También se encuentran que los lugares más frecuencia de viajes cambian cada hora y cada día.

Mediante clustering, el sistema computa las agrupaciones más lucrativas para cada hora y día de la semana, dando una puntuación a cada uno de los grupos basado en las correlaciones encontradas. El sistema se ejecuta varias veces sobre un clúster Spark, buscando la configuración más óptima.

Los resultados se guardan en una base de datos geoespacial, y puede consultarse mediante una aplicación web introduciendo la hora, día de la semana, y ubicación. El sistema recomienda las diez ubicaciones más cercanas, ordenadas por beneficio.

El sistema puede ser interesante para los Taxistas Amarillos de Nueva York, como una forma de incrementar los beneficios influyendo en sus desplazamientos de forma directo, libertad que los servicios competidores como Uber y Cabify no ofrecen, ya que los objetivos de tales taxistas les son fijados por la compañía.

Palabras clave: big data, data mining, Spark, spatial clustering, python, taxi

Abstract

This document proposes a system to maximize the income of taxi drivers, recommending the areas of the city with more frequency of trips and closer to the position of the taxi driver at the time of the query.

Analyzing 10M traces of trips made by the New York Yellow Taxis with a Spark cluster, we found correlations between the benefits and the time, distance and number of passengers. We also found that more frequent travel places change every hour and every day.

Through clustering, the system computes the most profitable groups for each hour and day of the week, giving a score to each of the groups based on the correlations found. The system runs several times on a Spark cluster, looking for the most optimal configuration.

The results are stored in a geospatial database, and can be viewed through a web application by entering the time, day of the week, and location. The system recommends the top ten closest locations, sorted by profit.

The system may be of interest to the Yellow Taxi drivers in New York, as a way to increase profits by influencing their wandering, something that competing services like Uber and Cabify do not offer, since the objectives of such taxi drivers are fixed by the company.

Keywords: big data, data mining, Spark, spatial clustering, python, taxi

Índice general

1. CAPÍTULO 1: INTRODUCCIÓN Y OBJETIVOS 3
1.1 Introducción 3
1.2 Objetivos 4
1.3 Estructura de la memoria 5
2. CAPÍTULO 2: ESTADO DEL ARTE 7
2.1 Recomendadores para Taxistas 7
2.2 Big Data 10
2.3 Computación Distribuida 14
2.4 Aprendizaje Automático 27
2.5 Base de Datos 33
2.6 Aplicaciones Web 37
2.7 Internet Hosting 41
3. CAPÍTULO 3: ANÁLISIS 43
3.1 Requisitos de usuario 43
3.2 Requisitos de software 49
3.3 Matriz de trazabilidad 54
4. CAPÍTULO 4: DISEÑO 55
4.1 Introducción 55
4.2 Arquitectura 55
4.3 Componentes 57
5. CAPÍTULO 5: IMPLEMENTACIÓN 60
5.1 Introducción 60
5.2 Estudio de los datos 60
5.3 Procesado de los datos 71
5.4 Modelo recomendador 72
5.5 Aplicación Spark 76
5.6 Base de datos espacial 77
5.7 Aplicación Web 78
6. CAPÍTULO 6: PRUEBAS Y RESULTADOS 80

6.1 Pruebas funcionales en PC Portátil 80
6.2 Pruebas de rendimiento en clúster universitario 81
7. CAPÍTULO 7: CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO 88
7.1 Conclusiones 88
7.2 Futuras líneas de trabajo 88
8. CAPÍTULO 8: PLANIFICACIÓN 91
8.1 Fases de desarrollo 91
9. CAPÍTULO 9: PRESUPUESTO 93
9.1 Medios Empleados 93
9.2 Presupuesto del trabajo 94
10. CAPÍTULO 10: ANEXOS 97
Anexo A: Capturas Aplicación Web 98
Anexo B: Propiedades completas de las trazas 101
Anexo C: Código 102
Anexo D: Normativa y Marco regulador 104
Referencias 105

Índice de figuras

Figura 1. Esquema general del sistema big data desarrollado. 4
Figura 2: Aplicación T-Finder usada en las pruebas de campo (Windows Phone 7) 8
Figura 3: Ejemplo de grafo de rutas a optimizar en el estudio 9
Figura 4: Ejemplo de ruta en *hubcap* 10
Figura 5: Big Data V's en los últimos años 12
Figura 6: Red de patologías encontrada por la Universidad de Chicago 13
Figura 7: Ejemplo de Web como sistema distribuido 16
Figura 8: Modelo cliente servidor 17
Figura 9: Peticiones y respuestas en modelo cliente servidor 17
Figura 10: Internet y el modelo cliente-servidor 18
Figura 11: Modelo P2P 19
Figura 12: Ejemplo de computación en *grid* 20
Figura 13: Computación en clúster 21
Figura 14: Map Reduce 22
Figura 15: Diagrama de Flujo de Hadoop 23
Figura 16: Diagrama de flujo de Spark 24
Figura 17: Ejemplo de trabajo con RDD 24
Figura 18: Ejecución de etapas 25
Figura 19: Planificación en Spark 26
Figura 20: Librerías de Spark 27
Figura 21: Ejemplo de DBSCAN 28
Figura 22: Diagrama de flujo K-means 29
Figura 23: Ejemplo de K-means 30
Figura 24: Mezcla de distribuciones gaussianas 32
Figura 25: Ejemplo de Gaussian Mixture Model 32
Figura 26: Ejemplo de base de datos relacional 35
Figura 27: Ejemplo de consulta PostGIS 37
Figura 28: Capas del modelo OSI 38
Figura 29: Modelo - Vista - Controlador 39
Figura 30: Ejemplo de integración con Google Maps API 40
Figura 31. Visión general del sistema 55
Figura 32: Apache Spark en modo *Standalone* 56
Figura 33: Arquitectura de la base de datos y aplicación web 57
Figura 34: Componentes del Modelo Recomendador 58
Figura 35: Componentes de la Transferencia a la base de datos 59
Figura 36: Componentes de la Consulta de los resultados 59
Figura 37: Distritos de New York City 61
Figura 38. Correlacciones con el pago total 63
Figura 39. Correlación del rendimiento 63
Figura 40. Correlación del dinero total producido 64
Figura 41. Diagrama de dispersión de coordenadas iniciales 66

Figura 42. Detalle del diagrama de dispersión de coordenadas iniciales 66
Figura 43. Frecuencia de viajes por hora del día 67
Figura 44. Clústeres obtenidos por franja horaria..... 70
Figura 45. Distritos en GeoJSON 72
Figura 46: Ejemplo del resultado del algoritmo *Quickhull*..... 73
Figura 47: Función de puntuación de recomendación 74
Figura 48. Polígonos de los clusters con mayor puntuación..... 75
Figura 49: SQL de la tabla "taxi_recommendation" 77
Figura 50: Consulta de recomendaciones a la base de datos 79
Figura 51: Comando de ejecución del sistema en el PC portátil 80
Figura 52: Comando de carga de resultados 80
Figura 53. Aplicación web tras la carga de las recomendaciones computadas 80
Figura 54: Comando de ejecución de pruebas de rendimiento de GMM 81
Figura 55: Tiempos de cálculo de clústers con GMM 82
Figura 56: Velocidad de cálculo de clústers con GMM 83
Figura 57: Comando de configuración del clúster 84
Figura 58: Comando de arranque del clúster 84
Figura 59: Comando de ejecución de aplicación en el clúster 84
Figura 60: Comando de parada del clúster 85
Figura 61: Captura de la UI de administración del cluster Spark 85
Figura 62. Tiempo de lectura del fichero en el clúster 86
Figura 63. Tiempo de computación de recomendaciones en el clúster 87
Figura 64. Polinomio de grado 9 y curva agregada de tiempo 89
Figura 65. Página principal 98
Figura 66. Trayecto en Google Maps 99
Figura 67. Administración de recomendaciones: listado 99
Figura 68. Administración de recomendaciones: edición 100

Índice de tablas

Tabla 1: RU-C-01 45
Tabla 2: RU-C-02 45
Tabla 3: RU-C-03 45
Tabla 4: RU-C-04 45
Tabla 5: RU-C-05 46
Tabla 6: RU-C-06 46
Tabla 7: RU-C-07 46
Tabla 8: RU-C-08 46
Tabla 9: RU-C-09 47
Tabla 10: RU-C-10 47
Tabla 11: RU-R-01 47
Tabla 12: RU-R-02 47
Tabla 13: RU-R-03 48
Tabla 14: RU-R-04 48
Tabla 15: RU-R-05 48
Tabla 16: RS-F-01 50
Tabla 17: RS-F-02 50
Tabla 18: RS-F-03 50
Tabla 19: RS-F-04 50
Tabla 20: RS-F-05 51
Tabla 21: RS-F-06 51
Tabla 22: RS-F-07 51
Tabla 23: RS-F-08 51
Tabla 24: RS-F-09 52
Tabla 25: RS-F-10 52
Tabla 26: RS-F-11 52
Tabla 27: RS-NF-R-01 52
Tabla 28: RS-NF-R-02 53
Tabla 29: RS-NF-R-03 53
Tabla 30: RS-NF-R-04 53
Tabla 31: RS-NF-D-01 53
Tabla 32: Correlación de frecuencias entre días de la semana 68
Tabla 33: Grupos de horas analizados 68
Tabla 34: Orden de los campos del resultado serializados 76
Tabla 35: Campos de la tabla “taxi_recommendation” 77
Tabla 36: Tiempos de cálculo de clústers con GMM 82
Tabla 37: Velocidad de cálculo de clústers con GMM 83
Tabla 38: Horas de desarrollo 95
Tabla 39: Coste de desarrollo 95
Tabla 40: Coste de material 95
Tabla 41: Otros costes 95
Tabla 42: Coste total 96
Tabla 43: Coste total con impuestos indirectos 96
Tabla 44. Propiedades completas de las trazas 101

Capítulo 1: Introducción y objetivos

1.1 Introducción

El negocio del servicio de transporte privado en las ciudades es cada vez más competitivo. En la actualidad se puede encontrar un buen número de alternativas al tradicional taxi. Compañías como Cabify, Uber o Lyft compiten por sustituir los taxis por un sistema de economía colaborativa, donde la tecnología juega un papel vital.

En la ciudad de Nueva York, los taxis tradicionales ya son superados en número por estos servicios en una proporción de cuatro a uno en 2016[1]. Una de las principales diferencias entre estos servicios, especialmente en ciudades como New York, se basa en que los usuarios consumen los servicios de transporte privado de diferentes maneras. En el caso de los servicios colaborativos, el usuario proporciona su ubicación y la tecnología hace el resto para seleccionar al conductor más apropiado en función de la distancia, valoración y disponibilidad a dicha ubicación. En el caso de los taxis, el usuario rara vez pacta el servicio dando su ubicación, simplemente juega con el factor "suerte" parando algún taxi que esté en la zona.

En este proyecto se pretende poner la "suerte" a favor del usuario de taxis proporcionando la información necesaria a los taxistas para estar en el lugar adecuado en el momento adecuado.

En este proyecto se realizará un sistema que ayude a los taxis amarillos de Nueva York a ser más competitivos. La NYC Taxi & Limousine Commission publica regularmente un dataset con trazas de millones de viajes realizados mensualmente en la ciudad de Nueva York[3]. Con un sistema *big data* podemos analizar qué características se correlacionan con un mayor beneficio económico para los taxistas, y ver qué lugares son más lucrativos en diferentes momentos del día y de la semana.

Cada vez es más fácil procesar grandes volúmenes de datos. Aplicaciones de código abierto de computación distribuida como Spark son cada vez más populares, a la vez que maduras. Hay tecnologías que permiten desplegar un clúster en minutos[2].

Teniendo en cuenta estos dos factores, computaremos las zonas más lucrativas para cada hora y día de la semana usando un sistema *big data*, y guardaremos los resultados obtenidos en una base de datos geoespacial. Para que los taxistas puedan consultarlo online, crearemos una aplicación web que acceda a ella, de tal forma que los taxistas puedan consultar qué lugares son más recomendables para su ubicación, día de la semana y momento del día. Además, la integraremos con Google Maps para que, al escoger el lugar de preferencia, comience la navegación guiada hacia dicha zona.

De esta manera pretendemos dar una ventaja cuantitativa a los conductores de taxis. Si bien los usuarios de Uber o Cabify solo tienen que abrir una aplicación móvil para conseguir un conductor, pretendemos que los usuarios de taxi sólo tengan que levantar la

mano ya que los conductores sabrán donde es más probable que sus servicios sean solicitados.

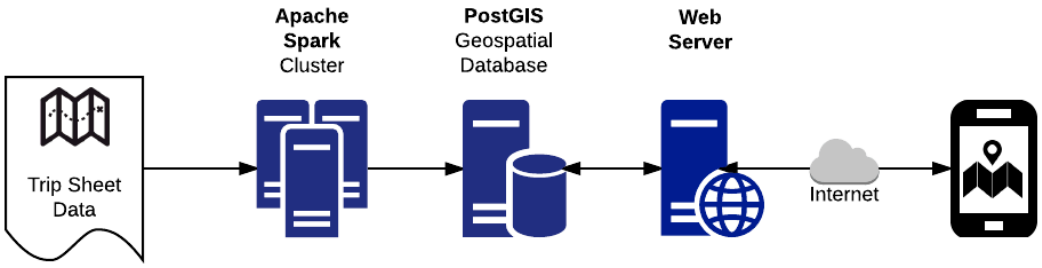


Figura 1. Esquema general del sistema big data desarrollado.

1.2 Objetivos

El objetivo fundamental de este proyecto es el de crear un sistema que ayude a mejorar el beneficio económico de los conductores de taxis mediante recomendaciones de lugares más lucrativos, analizando el popular dataset de trazas de viajes ofrecido por NYC Taxi & Limousine Commision[3]. En base a ese objetivo principal, se proponen los siguientes objetivos parciales:

- Estudio del contexto y la motivación que llevan a la realización del proyecto.
- Estudio de las tecnologías que van a ser utilizadas en el desarrollo del sistema.
- Estudio de las trazas de los taxis.
- Diseño de un sistema big data que se adapte a las necesidades del proyecto.
- Implementación del sistema en base al diseño del mismo.
- Visualización de los resultados del procesamiento de los datos y de la aplicación desarrollada.
- Realización de pruebas de rendimiento del sistema en diferentes entornos.
- Extracción de conclusiones del desarrollo del proyecto.
- Planteamiento de líneas futuras ofreciendo continuidad al proyecto.

1.3 Estructura de la memoria

Para poder facilitar la estructura de la memoria se detalla a continuación una estructura de la misma:

- Capítulo 1: Introducción y objetivos.
  Motivaciones del proyecto y establecimiento de los objetivos del mismo recogiendo la idea general del trabajo que se va a realizar. Además, se enuncia la estructura de la memoria.

- Capítulo 2: Estado del arte.
  Detalle de las tecnologías utilizadas en el desarrollo del proyecto las cuales deben ser conocidas por el lector previamente a la exposición de la solución técnica.

- Capítulo 3: Análisis
  Analizaremos los requisitos del sistema, nos familiarizamos con el dataset y buscamos correlacciones.

- Capítulo 4: Diseño
  Pasos que se han seguido en el desarrollo del sistema diseñado, desde la carga inicial de datos hasta la visualización de los resultados obtenidos en el procesado.

- Capítulo 5: Implementación
  Se describe el proceso de implementación y los pasos necesarios para su uso.

- Capítulo 6: Pruebas y resultados.
  Pruebas que se han realizado para conocer los límites del entorno y los tiempos de ejecución alcanzados.

- Capítulo 7: Conclusiones y futuras líneas de trabajo.
  Análisis de la consecución de los objetivos y las posibles líneas futuras de trabajo a realizar.

- Capítulo 8: Planificación.
  Fases de desarrollo del proyecto y su diagrama de Gantt correspondiente.

- Capítulo 9: Presupuesto.
  En este capítulo se exponen los recursos empleados y el presupuesto para la realización del proyecto.

- Capítulo 10: Anexos.
  - Anexo A: Capturas Aplicación Web.
  - Anexo B: Propiedades completas de las trazas.
  - Anexo C: Código
  - Anexo D: Normativa y marco regulador.

- Referencias

Capítulo 2: Estado del Arte

2.1 Recomendadores para Taxistas

2.1.1 T-Finder

En 2012, la revista IEEE Transactions on Knowledge and Data Engineering publicó "T-Finder: A Recommender System for Finding Passengers and Vacant Taxis"[4]. El objetivo es el más parecido que he encontrado a este proyecto.

En dicho estudio se presentó un sistema de recomendación tanto para los taxistas como para las personas que esperan tomar un taxi, utilizando el conocimiento de: 1) los patrones de movilidad de los pasajeros y 2) los comportamientos de recogida / abandono de los taxistas aprendidos de las trayectorias GPS de los taxis.

En primer lugar, este sistema de recomendación proporciona a los taxistas algunas ubicaciones y las rutas a estos lugares, hacia los cuales es más probable que recojan pasajeros rápidamente (durante las rutas o en estos lugares) y maximicen los beneficios del próximo viaje. En segundo lugar, recomienda a las personas con algunas ubicaciones (a poca distancia) donde pueden encontrar fácilmente taxis vacantes.

Analizan las trayectorias GPS de los taxis utilizando un modelo probabilístico que estima el beneficio de las ubicaciones candidatas para un conductor particular basado en donde y cuando el conductor solicita la recomendación. Se basaron en trayectorias históricas generadas por más de 12.000 taxis durante 110 días y validaron el sistema con evaluaciones extensivas incluyendo estudios de campo. Desarrollaron una aplicación móvil para el sistema.

Es el sistema más parecido al proyecto en el que he trabajado que he encontrado, pero es mucho más sofisticado porque calculan trayectos, y además lo validaron con un estudio de campo. En la Figura 2 podemos ver la aplicación que desarrollaron.

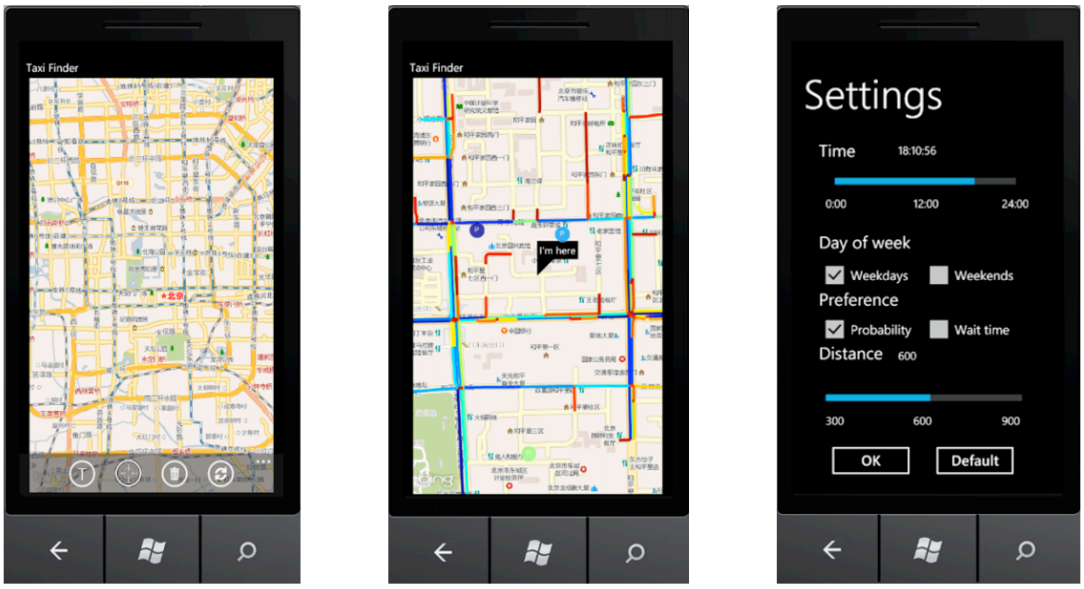


Figura 2: Aplicación T-Finder usada en las pruebas de campo (Windows Phone 7)

2.1.2 A cost-effective recommender system for taxi drivers

Se trata de un estudio publicado en 2014 en la revista KDD '14[5].

El paper propone desarrollar un sistema de recomendación rentable para los taxistas. El sistema se diseña para maximizar sus ganancias al seguir las rutas recomendadas para encontrar pasajeros. En concreto, diseñan primero una función de objetivo de beneficio neto para evaluar los beneficios potenciales de las rutas de conducción. A continuación, desarrollan una representación gráfica de las redes de carreteras mediante la extracción de los rastreos históricos GPS de taxis y proporcionan una estrategia de fuerza bruta para generar una ruta de conducción óptima para la recomendación.

Sin embargo, un desafío crítico a lo largo de esta línea es el alto costo computacional. Teniendo esto en cuenta, desarrollan una nueva estrategia de recursión basada en la función de beneficio neto para buscar rutas candidatas óptimas de manera eficiente. En particular, en lugar de recomendar una secuencia de puntos de recogida y dejar al conductor decidir cómo llegar a esos puntos, el sistema recomendador es capaz de proporcionar una ruta de conducción completa, con las direcciones a seguir. De esta forma, los conductores pueden encontrar más clientes potenciales, incrementando el beneficio total.

Esto hace que el sistema de recomendación sea más práctico y rentable que otros sistemas de recomendación existentes. Finalmente, realizan experimentos en un conjunto de datos del mundo real recolectados en el área de la Bahía de San Francisco. Los resultados experimentales validan claramente la efectividad del sistema de recomendación propuesto.

La diferencia principal es que buscan dar la ruta más óptima, así como mejorar la eficiencia con la encadenación de viajes, cuando en este proyecto se abstrae de dicho cálculo. En la Figura 3 vemos un ejemplo del cálculo de dichas rutas.

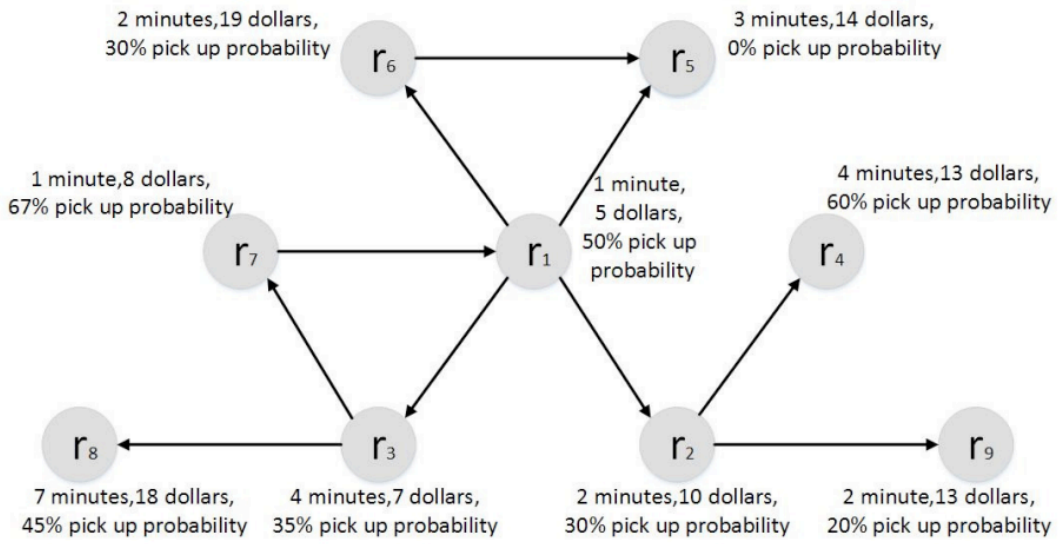


Figura 3: Ejemplo de grafo de rutas a optimizar en el estudio

2.1.3 hubcab

hubcab[6] es una aplicación web que permite investigar exactamente cómo y cuándo los taxis recogen o dejan a los individuos en la ciudad de Nueva York. El sistema visualiza agrupadas las zonas de recogida y bajada del taxi. Se pueden ver los lugares donde sus viajes de taxi comienzan y terminan y para descubrir qué número de personas en esa área siguen los mismos patrones de viaje.

Su objetivo principal es encontrar patrones que permitan ayudar a compartir taxis para reducir la huella de carbono. Es un poco diferente de este proyecto, pero el procesamiento de los datos y construcción del modelo es interesante.

Analizaron 170 millones de viajes en taxi de los taxis amarillos en la ciudad de Nueva York en 2011. Usando OpenStreetMap y Python, las calles fueron cortadas en más de 200.000 segmentos de 40m de longitud. Los puntos de recogida y devolución se compararon con los segmentos de calle más cercanos. Se excluyeron tipos de calles que difícilmente pudieran tener recogidas o bajadas de taxi, tales como senderos, troncos, carreteras de servicio, etc. Al hacer zoom, aparecen líneas amarillas y azules que representan puntos de recogida y devolución respectivamente están en una escala logarítmica. En niveles de zoom más altos, se representan como puntos, y fueron generados a través de un script en Arcpy. La base de datos es MongoDB, y contiene todos los segmentos de calle y sus coordenadas, así como los flujos entre ellos. El

número de todos los pares de segmentos de calle posibles es de más de 40 mil millones por mapa.

En la Figura 4 podemos ver un ejemplo. Es una captura de pantalla de la aplicación, en la cual se ha consultado cuán frecuente es la ruta desde el punto amarillo al punto azul. Podemos navegar en diferentes puntos del día con la rueda superior derecha, y podemos ver los beneficios obtenidos de compartir coche en el cuadro verde superior izquierda.

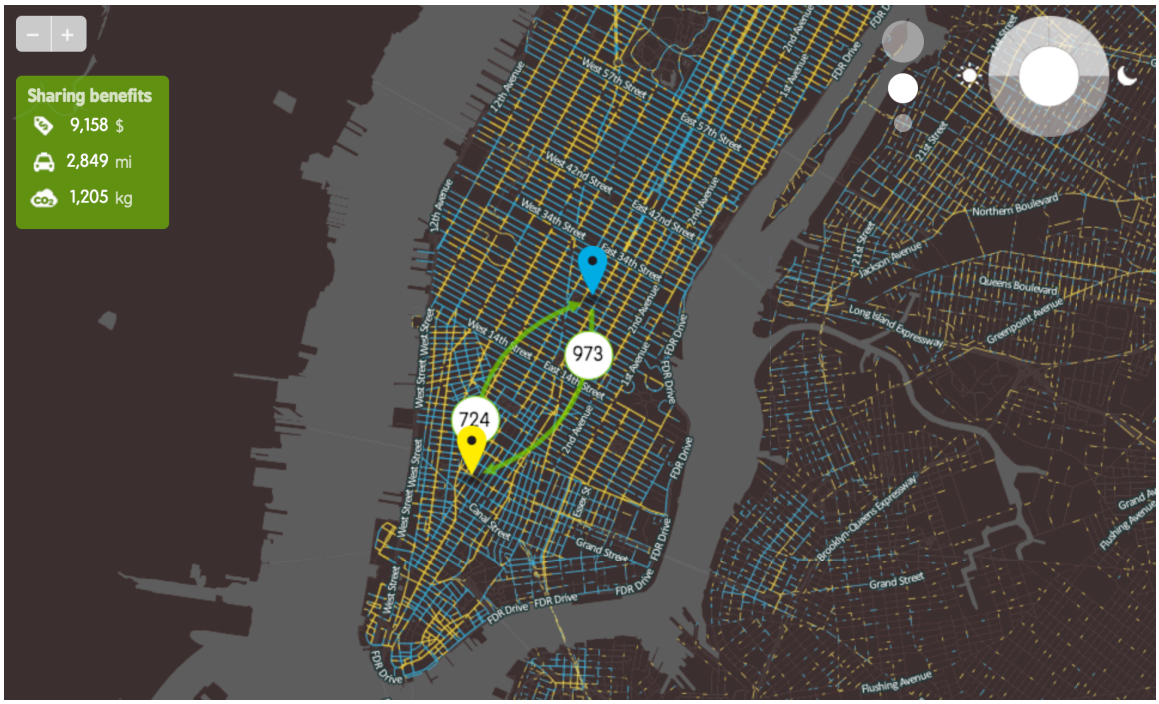


Figura 4: Ejemplo de ruta en *hubcap*

2.2 Big Data

2.2.1 Introducción

Big Data es un término muy amplio, pero en general se usa para referirse a conjuntos de datos tan grandes que son difíciles de almacenar y procesar con una computadora convencional

El concepto Big Data ha existido desde hace años. El término comenzó a popularizarse en los años noventa, gracias especialmente a una presentación que realizó John Mashey en 1998[7]. Hoy en día la mayoría de las organizaciones saben que si captan todos los datos que fluyen en sus negocios, pueden realizar análisis sobre ellos y obtener un valor significativo. Pero incluso en los años cincuenta, décadas antes de que alguien pronunciara el término Big Data, las empresas ya realizaban análisis rudimentarios para descubrir ideas y tendencias.

Si bien el problema de trabajar con datos que exceden el poder de computación o el almacenamiento de una sola computadora no es nuevo, la omnipresencia, la escala y el valor de este tipo de computación se ha expandido mucho en los últimos años.

En 2017, se crean cada día 2.5 quintillones de bytes de datos. Para ponerlo en perspectiva, el 90 por ciento de los datos en el mundo de hoy se ha creado sólo en los últimos dos años. Y teniendo en cuenta los nuevos dispositivos, sensores y tecnologías emergentes, el crecimiento de datos probablemente acelerará aún más.[8]

El aumento de la cantidad de datos disponibles presenta nuevas oportunidades y nuevos problemas. En general, tener más datos sobre los clientes (y los clientes potenciales) debería permitir a las empresas adaptar mejor sus productos y esfuerzos de marketing para mejorar el nivel de satisfacción y el negocio. Las empresas que son capaces de recopilar gran cantidad de datos tienen la oportunidad de llevar a cabo un análisis más profundo y más rico.

Mientras que mejorar el análisis es positivo, el Big Data también pueden crear saturación y ruido. Las empresas deben ser capaces de manejar mayores volúmenes de datos, al mismo tiempo que identifican e ignoran el ruido. Determinar qué hace que los datos sean relevantes se convierte en un factor clave. Los datos estructurados, que consisten en valores numéricos, pueden almacenarse y clasificarse fácilmente. Los datos no estructurados, como correos electrónicos, videos y documentos de texto, pueden requerir técnicas más sofisticadas para que se apliquen antes de que sea útil. El concepto Big Data engloba todos ellos.[9]

2.2.2 Definición

Big Data es un término para conjuntos de datos que son tan grandes o complejos que el software y el hardware convencional de procesamiento de datos es inadecuado para tratar con ellos. Esto incluye la captura, almacenamiento, análisis, consulta, compartición, transferencia, visualización, actualización y privacidad de la información.[10]

En un informe de investigación de 2001[11] y en conferencias relacionadas, META Group (ahora Gartner) definió los desafíos y oportunidades de crecimiento de datos en tres dimensiones:

- Volumen (cantidad de datos)
- Velocidad (velocidad de entrada y salida de datos)
- Variedad (tipos de datos y fuentes)

Gartner, y ahora gran parte de la industria, siguen utilizando este modelo de "3Vs" para describir grandes datos. En 2012, Gartner actualizó su definición de la siguiente manera: "Big Data es un activo de información de gran volumen, alta velocidad y/o alta variedad que exige formas innovadoras de procesamiento de información rentables que permiten mejorar el conocimiento, la toma de decisiones y los procesos de automatización".

La definición de Gartner de las 3Vs sigue siendo ampliamente utilizada, y se da una definición consensuada en la industria que establece que "Big Data representa los activos de información caracterizados por un alto volumen, velocidad y variedad, que requieren de tecnología y métodos analíticos específicos para su transformación en valor".[12]

Por otro lado, algunas organizaciones han añadido una nueva V, “Veracidad”, que es la certeza de que los datos contenidos en el sistema son reales. Otras, “Volatilidad”, “Viabilidad”, y “Variabilidad”. En la Figura 5 podemos ver un resumen de las diferentes V a lo largo de los últimos años.[13]

Tal diversidad ha convertido a Big Data en un término paraguas, el cual podemos decir que engloba aquellas estrategias y tecnologías no tradicionales que se necesitan para reunir, organizar, procesar y recopilar información de grandes conjuntos de datos

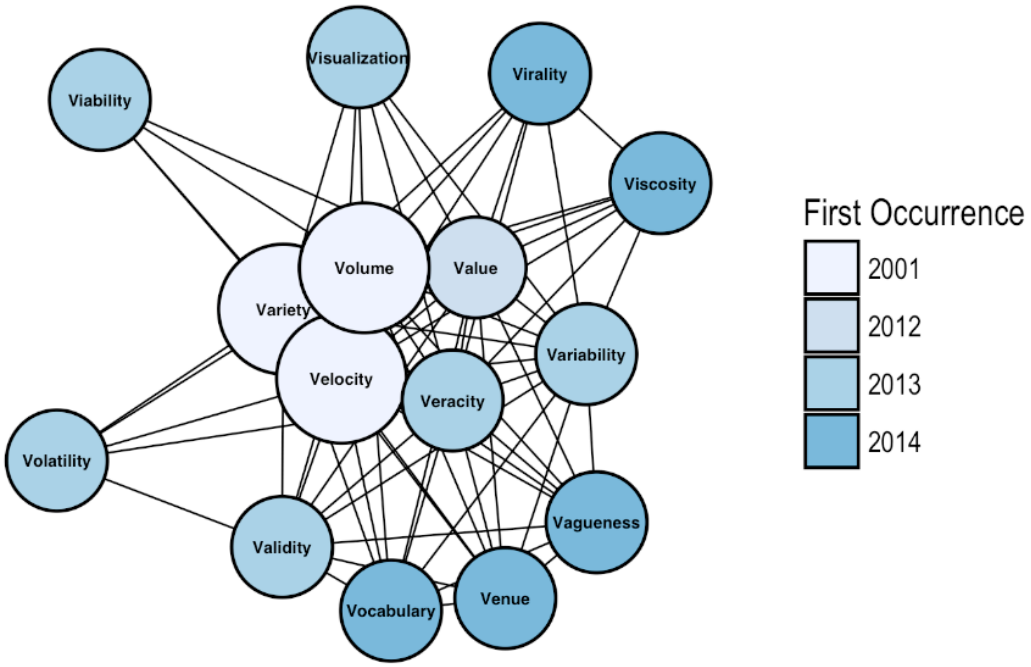


Figura 5: Big Data V's en los últimos años

2.2.3 Minería de Datos

La minería de datos es el proceso computacional de descubrir patrones en grandes conjuntos de datos con ayuda de aprendizaje automático, estadística, y sistemas de bases de datos[14]. El objetivo general del proceso de minería de datos es extraer información de un conjunto de datos y transformarla en una estructura comprensible para uso posterior. Es una disciplina clave para extraer valor en los sitemas Big Data.

Aparte del paso de análisis de los datos en bruto, la minería de datos involucra otras tareas:

- Administración de bases de datos
- Preprocesamiento de datos
- Modelos e inferencias estadísticas
- Extracción de métricas relevantes
- Complejidad computacional
- Postprocesamiento de las estructuras encontradas
- Visualización de los resultados
- Actualización de los datos

La minería de datos tiene múltiples aplicaciones. Es aplicada en marketing, juegos de mesa, ciencia, ingeniería, entre otras áreas. En la Figura 6 podemos ver un grafo realizado en un estudio de 2011 por la Universidad de Chicago[15], en el que usaron minería de datos para analizar 1.5 millones de registros de pacientes con diferentes desórdenes mentales. Los resultados mostraron que un gran número de personas tenían múltiples patologías, y ayudaron a detectar patrones de factores ambientales que podían influir en ellas.

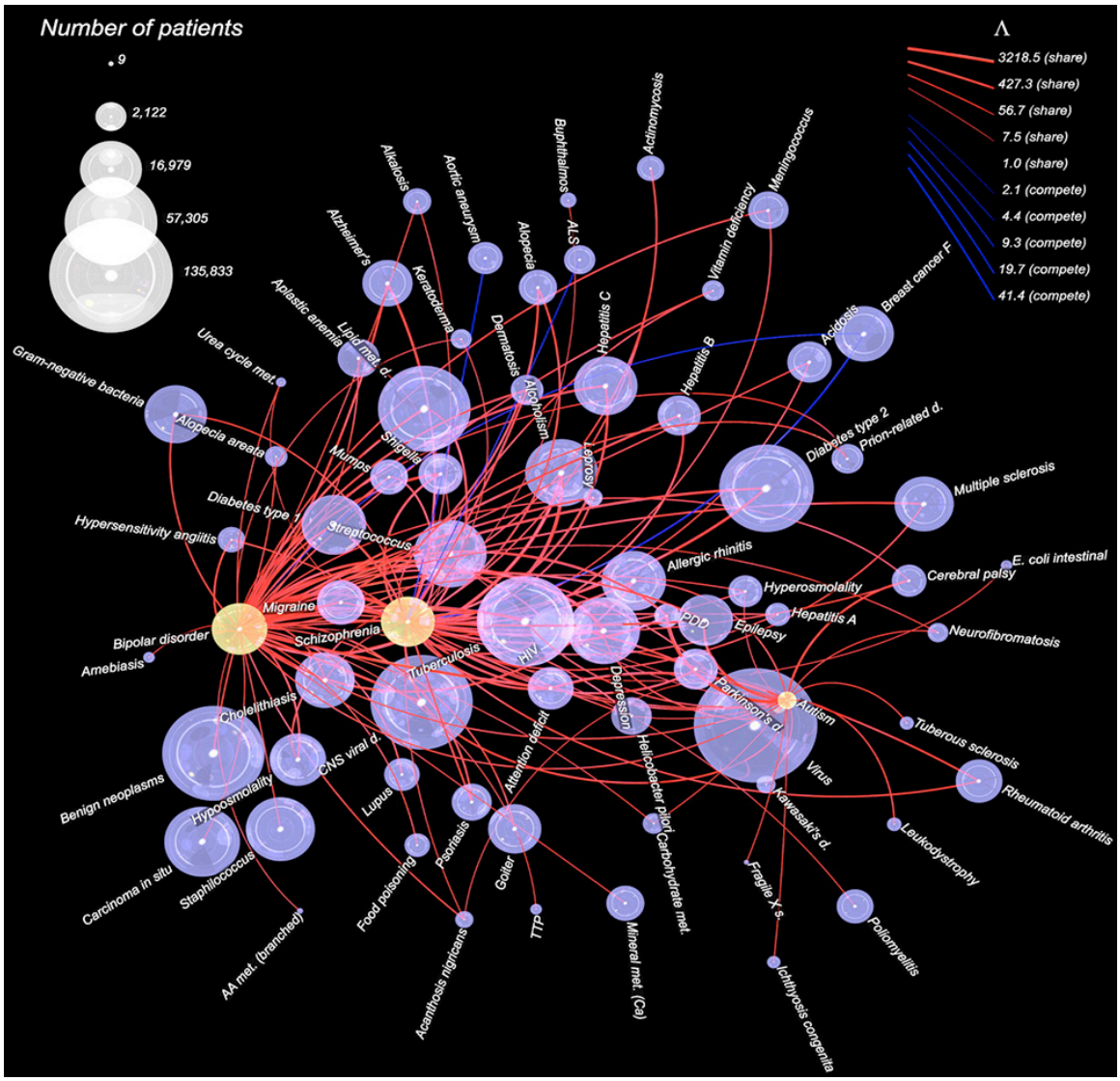


Figura 6: Red de patologías encontrada por la Universidad de Chicago

Podemos dividir el trabajo de minería de datos en las siguientes fases:

1. Definición del problema: conocer el dominio y especificar las necesidades que pretendemos resolver.

2. Análisis de los datos: Estudiar los datos, ver qué representan, cuál es su origen y formato.

3. Preprocesado de los datos: Limpiar los datos de valores incorrectos y organizarlos de forma que puedan ser utilizados.

4. Modelación: Diseñar los algoritmos que van a ser utilizados para buscar los patrones en los datos.

5. Validación: Debemos comprobar que los resultados resuelven los requisitos especificados.

El paso final del descubrimiento de conocimiento a partir de datos es verificar que los patrones producidos por los algoritmos de minería de datos se dan en otros conjuntos de datos. No todos los patrones encontrados por los algoritmos de minería de datos son necesariamente válidos. Es común que los algoritmos de minería de datos encuentren patrones en el conjunto de entrenamiento que no están presentes en el conjunto de datos generales, lo que se conoce como *overfitting*.

Para superar esto, la evaluación utiliza un subcojunto de los datos como datos de validación, y dicho conjunto no es incluido en el entrenamiento. Los patrones aprendidos se aplican a este conjunto de pruebas, y la salida resultante se compara con la salida deseada.

Por ejemplo, un algoritmo de minería de datos que trataba de distinguir *spam* de correos electrónicos legítimos sería entrenado en un subconjunto extraído para el entrenamiento. Una vez entrenados, los patrones aprendidos serían aplicados al subconjunto determinado para pruebas, que no han sido incluidos en el subconjunto de entrenamiento. La precisión de los patrones se puede medir a partir de cuántos correos electrónicos se clasifican correctamente.

Si los patrones aprendidos no cumplen con los estándares deseados, es necesario reevaluar y cambiar los pasos de pre-procesamiento y de minería de datos. Si los patrones aprendidos cumplen con los estándares deseados, entonces el paso final es interpretar los patrones aprendidos y convertirlos en conocimiento.

2.3 Computación Distribuida

Un sistema distribuido consiste en una colección de máquinas autónomas conectadas por una red de comunicación y equipadas con sistemas de software diseñadas para producir un sistema de computación integrado y consistente. Los sistemas distribuidos permiten a las personas cooperar y coordinar sus actividades de forma más efectiva y eficiente[16].

Los objetivos clave de los sistemas distribuidos son los siguientes:

- Compartición de recursos: En un sistema distribuido, los recursos como hardware, software y datos pueden ser fácilmente compartidos entre los usuarios. Por ejemplo, se puede compartir una impresora entre un grupo de usuarios.

- Apertura: La apertura de los sistemas distribuidos se logra especificando interfaces de software clave del sistema y poniéndola a disposición de los desarrolladores, de tal forma que el sistema se puede ampliar.

- Concurrencia: La simultaneidad de procesamiento se puede conseguir enviando solicitudes a múltiples máquinas conectadas por redes al mismo tiempo, siendo las peticiones procesadas en paralelo.

- Escalabilidad: Un sistema distribuido que se ejecuta en un conjunto pequeño número de máquinas puede aumentar la potencia de procesamiento ampliando el número de máquinas.

- Tolerancia a fallos: Las máquinas conectadas por redes pueden considerarse como recursos redundantes. Un sistema de software puede instalarse en múltiples máquinas para que ante fallos de hardware o fallos de software, los errores puedan ser detectados y tolerados por otras máquinas.

- Podemos identificar ocho formas de transparencia. Éstas proveen un resumen útil de la motivación y metas de los sistemas distribuidos[17]:

    - Transparencia de Acceso: Permite el acceso a los objetos de información remotos de la misma forma que a los objetos de información locales.

    - Transparencia de Localización: Permite el acceso a los objetos de información sin conocimiento de su localización

    - Transparencia de Concurrencia: Permite que varios procesos operen concurrentemente utilizando objetos de información compartidos y de forma que no exista interferencia entre ellos.

    - Transparencia de Replicación: Permite utilizar múltiples instancias de los objetos de información para incrementar la fiabilidad y las prestaciones sin que los usuarios o los programas de aplicación tengan por que conoces la existencia de las replicas.

    - Transparencia de Fallos: Permite a los usuarios y programas de aplicación completar sus tareas a pesar de la ocurrencia de fallos en el hardware o en el software.

    - Transparencia de Migración: Permite el movimiento de objetos de información dentro de un sistema sin afectar a los usuarios o a los programas de aplicación.

    - Transparencia de Prestaciones. Permite que el sistema sea reconfigurado para mejorar las prestaciones mientras la carga varia.

- Transparencia de Escalado: Permite la expansión del sistema y de las aplicaciones sin cambiar la estructura del sistema o los algoritmos de la aplicación.

En la década de 1990 la popularidad de internet trajo a la informática la computación distribuida basada en la Web. Una base de esta forma de tratamiento de la información es la computación distribuida que se lleva a cabo en sistemas informáticos distribuidos. Estos sistemas comprenden los tres componentes fundamentales siguientes:

- Computadoras personales y computadoras de servidor potentes.
- Redes locales rápidas y de amplia extensión, Internet.
- Sistemas, en particular sistemas operativos distribuidos y software de aplicación.

En la figura podemos ver un ejemplo de la visión de Internet como un sistema de computación distribuida, con bases de datos, servidores de trabajo, servidores web, y clientes.

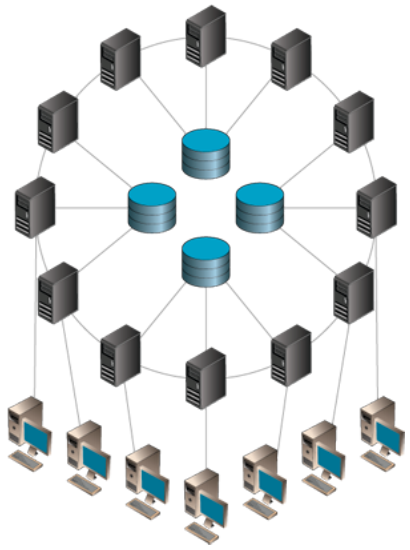


Figura 7: Ejemplo de Web como sistema distribuido

2.3.1 Cliente-Servidor

El modelo cliente-servidor es una estructura de aplicación distribuida que particiona tareas o cargas de trabajo entre los proveedores de un recurso o servicio, llamados *servidores*, y los solicitantes de servicio, llamados *clientes*[18].

Figura 8: Modelo cliente servidor

La arquitectura cliente-servidor de un sistema distribuido es el modelo más conocido y más ampliamente adoptado en la actualidad. Hay un conjunto de procesos servidores, cada uno actuando como un gestor de recursos para una colección de recursos de un tipo, y una colección de procesos clientes, cada uno llevando a cabo una tarea que requiere acceso a algunos recursos hardware y software compartidos. Los gestores de recursos a su vez podrían necesitar acceder a recursos compartidos manejados por otros procesos, así que algunos procesos son ambos clientes y servidores.

En el modelo, cliente-servidor, todos los recursos compartidos son mantenidos y manejados por los procesos servidores. Los procesos clientes realizan peticiones a los servidores cuando necesitan acceder a algún recurso. El servidor lleva a cabo la acción requerida y envía una respuesta al proceso cliente.

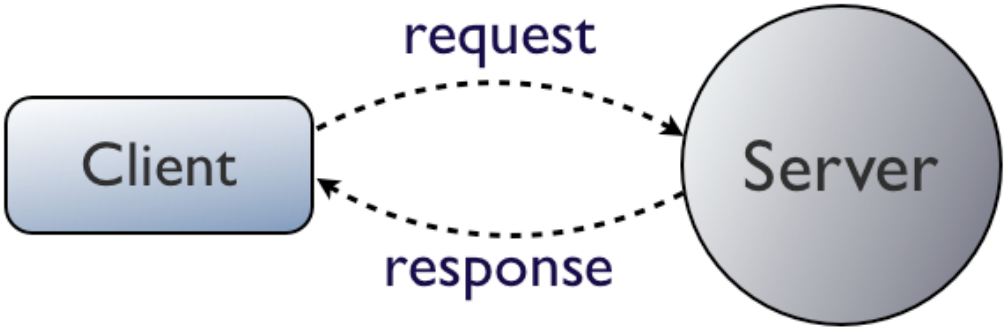


Figura 9: Peticiones y respuestas en modelo cliente servidor

Es una arquitectura especialmente popular en Internet. Las páginas web que visitamos a diario están construidas sobre dicho modelo. Los ordenadores de sobremesa, portátiles, Tabletas, móviles, todos acceden a las webs alojadas en internet siguiendo dicha arquitectura. En la Figura 10 podemos ver un ejemplo.

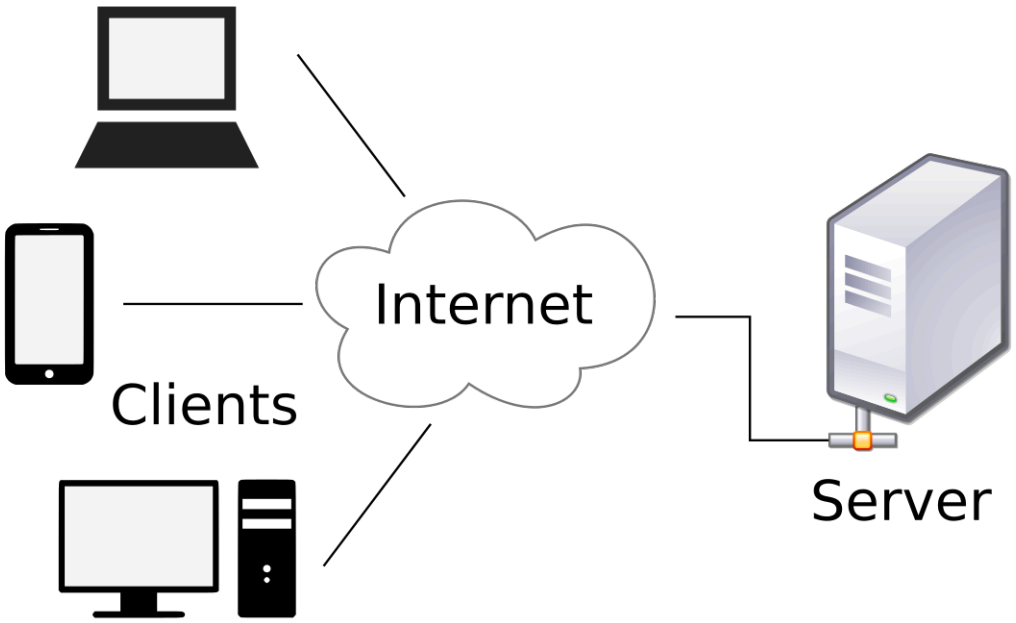


Figura 10: Internet y el modelo cliente-servidor

2.3.2 Peer to Peer

El modelo red de pares, en inglés *Peer to Peer* (P2P) es una arquitectura de aplicación distribuida que divide tareas o cargas de trabajo entre pares. Todos los nodos de la red son igualmente privilegiados, participantes equipotentes en la aplicación. Se dice que forman una red entre iguales.[19] En la figura podemos ver un ejemplo.

Una red peer-to-peer está diseñada en torno a la noción de que los nodos funcionan como clientes y servidores para los otros nodos de la red, simultáneamente. Este modelo de disposición de red difiere del modelo cliente-servidor en el que la comunicación suele hacerse desde y hacia un servidor central.

Figura 11: Modelo P2P

Podemos encontrar tres tipos de redes:

- Redes no estructuradas: Las redes peer-to-peer no estructuradas no imponen una estructura particular en la red de superposición por diseño, sino que son formadas por nodos que se conectan al azar entre sí. Debido a que no hay una estructura globalmente impuesta sobre ellos, las redes no estructuradas son fáciles de construir y permiten optimizaciones localizadas a diferentes regiones de la superposición

- Redes estructuradas: En las redes peer-to-peer estructuradas, la superposición se organiza en una topología específica y el protocolo asegura que cualquier nodo puede buscar eficientemente en la red un archivo o recurso, incluso si el recurso es extremadamente raro.

- Modelos híbridos: modelos híbridos son una combinación de modelos peer-to-peer y cliente-servidor. Un modelo híbrido común es tener un servidor central que ayude a los compañeros a encontrarse. Hay una gran variedad de modelos híbridos, todos los cuales hacen concesiones entre la funcionalidad centralizada proporcionada por una red estructurada de cliente-servidor y la igualdad de nodo proporcionada por las redes puramente peer-to-peer no estructuradas. En la actualidad, los modelos híbridos tienen un mejor rendimiento que las redes puras no estructuradas o las redes puras estructuradas porque ciertas funciones, como la búsqueda, requieren una funcionalidad centralizada, pero se benefician de la agregación descentralizada de nodos proporcionada por redes no estructuradas.

2.3.3 Grids

Computación en malla o rejilla, en inglés *grid computing*, es la colección de recursos informáticos en múltiples y distintas ubicaciones para alcanzar un objetivo común. La

rejilla puede considerarse como un sistema distribuido con cargas de trabajo no interactivas que involucran un gran número de archivos[20].

La computación en cuadrícula se distingue de los sistemas convencionales de computación de alto rendimiento, como la computación en clúster, en la que los ordenadores de cuadrícula tienen cada nodo configurado para realizar una tarea o aplicación diferente. Los ordenadores de cuadrícula también tienden a ser más heterogéneos y geográficamente dispersos que los ordenadores de clúster. Aunque una sola *grid* puede ser dedicada a una aplicación particular, normalmente se utiliza para una realizar diferentes tareas. Los tamaños de la rejilla pueden ser bastante grandes.

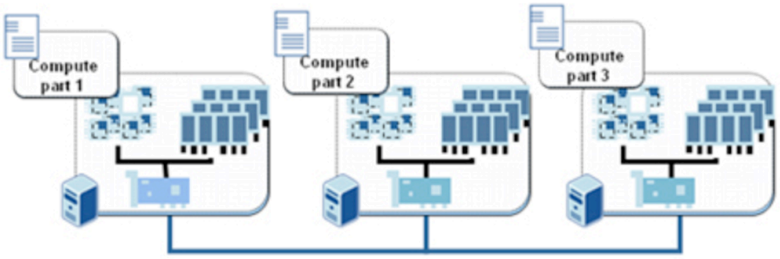


Figura 12: Ejemplo de computación en *grid*

2.3.4 Clústers

Un clúster de ordenadores consta de un conjunto de ordenadores conectados que trabajan juntos para que, en muchos aspectos, puedan ser vistos como un único sistema. A diferencia de la computación *grid*, los clústeres de ordenadores tienen cada nodo configurado para realizar la misma tarea, controlada y programada por el software, y suelen estar geolocalizados en el mismo sitio[21].

Los componentes de un clúster suelen estar conectados entre sí a través de redes de área local rápida, con cada nodo (ordenador utilizado como servidor) ejecutando su propia instancia de un sistema operativo. En la mayoría de los casos, todos los nodos utilizan el mismo hardware, el mismo sistema operativo, y el mismo software, aunque en algunas configuraciones se pueden utilizar diferentes sistemas operativos cada computadora o hardware diferente.

Por lo general se despliegan para mejorar el rendimiento y la disponibilidad sobre la de una sola computadora, mientras que por lo general son mucho más rentables que los equipos individuales de velocidad o disponibilidad comparable.

Los clusters surgieron como resultado de la convergencia de una serie de tendencias informáticas, incluyendo la disponibilidad de microprocesadores de bajo costo, redes de alta velocidad y software para computación distribuida de alto rendimiento

Tienen una amplia gama de aplicabilidad e implementación, que van desde los clusters de pequeñas empresas con un puñado de nodos a algunos de los supercomputadores más rápidos del mundo, como el Sequoia de IBM[22].

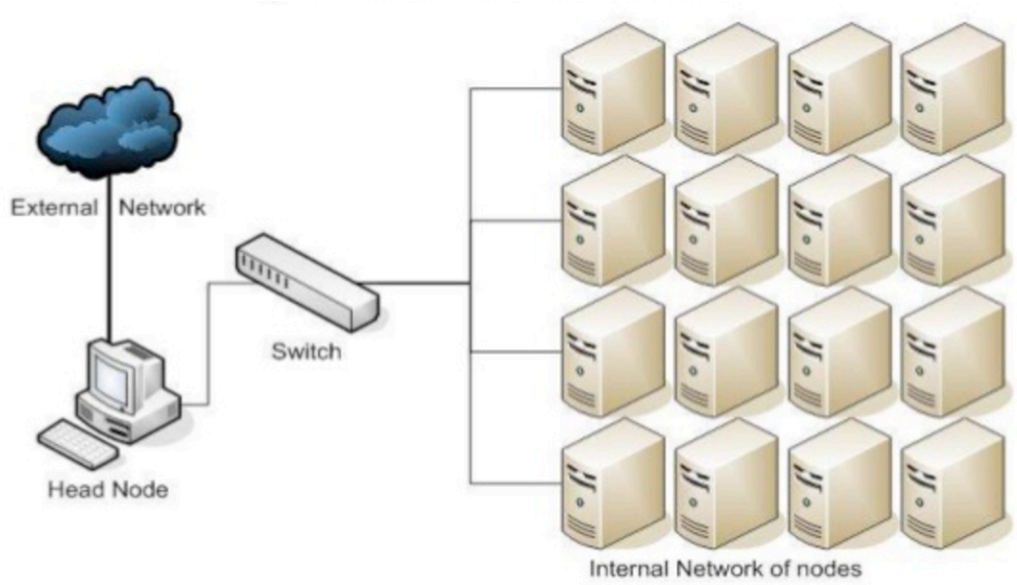


Figura 13: Computación en clúster

2.3.5 Map Reduce

MapReduce es un modelo de programación y una implementación asociada para procesar y generar grandes conjuntos de datos con un algoritmo paralelo distribuido en un clúster[23].

Se compone esencialmente de dos métodos diferentes:

- Map: Realiza el filtrado y la clasificación de la entrada, en forma de clave/valor.

- Reduce: Realiza una operación de resumen sobre los resultados computados por Map.

El sistema MapReduce orquesta el procesamiento de los datos sobre un sistema distribuido, ejecutando las distintas tareas en paralelo, gestionando todas las comunicaciones y transferencias de datos entre las distintas partes del sistema, y proporcionando redundancia y tolerancia a fallos.

Map y Reduce son funciones típicas utilizadas en la programación funcional, y el sistema se inspira en ellos, aunque su propósito no es el exactamente el mismo que en sus formas originales.

Los sistemas de hoy en día que implementan MapReduce se ocupan de los detalles de la partición de los datos de entrada, la orquestación de la ejecución del programa a través de un conjunto de máquinas, la tolerancia a fallos de las mismas, y la gestión de la comunicación entre ellas. Esto permite a los programadores sin experiencia con sistemas paralelos y distribuidos utilizar fácilmente los recursos de un sistema distribuido grande.

Los sistemas MapReduce suelen procesar siguiendo los siguientes pasos:

- Map: Cada nodo de trabajo aplica la función Map a los datos locales, y escribe la salida en un almacenamiento temporal. Un nodo maestro asegura que sólo se procese una copia de los datos de entrada redundantes.

- Redistribución: Los nodos redistribuyen los datos basados en los resultados del paso anterior, de modo que todos los valores pertenecientes a una misma clave se encuentren en el mismo nodo de trabajo.

- Reducir: Los nodos procesan ahora los grupos de datos de cada clave, en paralelo.

En la Figura 14 podemos ver un ejemplo del procedimiento en su conjunto

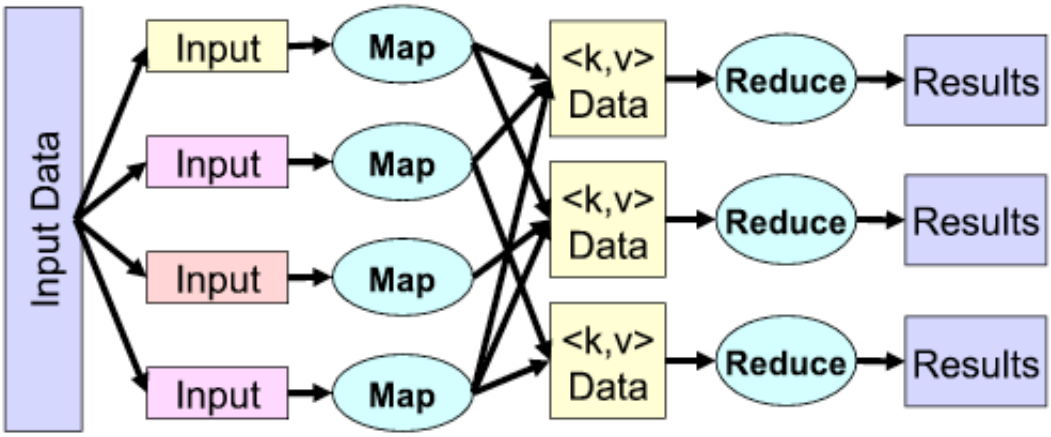


Figura 14: Map Reduce

2.3.6 Apache Hadoop

El núcleo de Apache Hadoop consiste en su modelo de almacenamiento, conocida como Hadoop Distributed File System (HDFS), y el procesado siguiendo el sistema MapReduce[24].

Hadoop divide los archivos en bloques grandes y los distribuye entre los nodos de un clúster. A continuación, transfiere el código empaquetado a los nodos para procesar los datos en paralelo. Este enfoque se aprovecha de la localidad de datos, donde los nodos manipulan los datos a los que tienen acceso. En la Figura 15 podemos ver un ejemplo de dicho proceso.

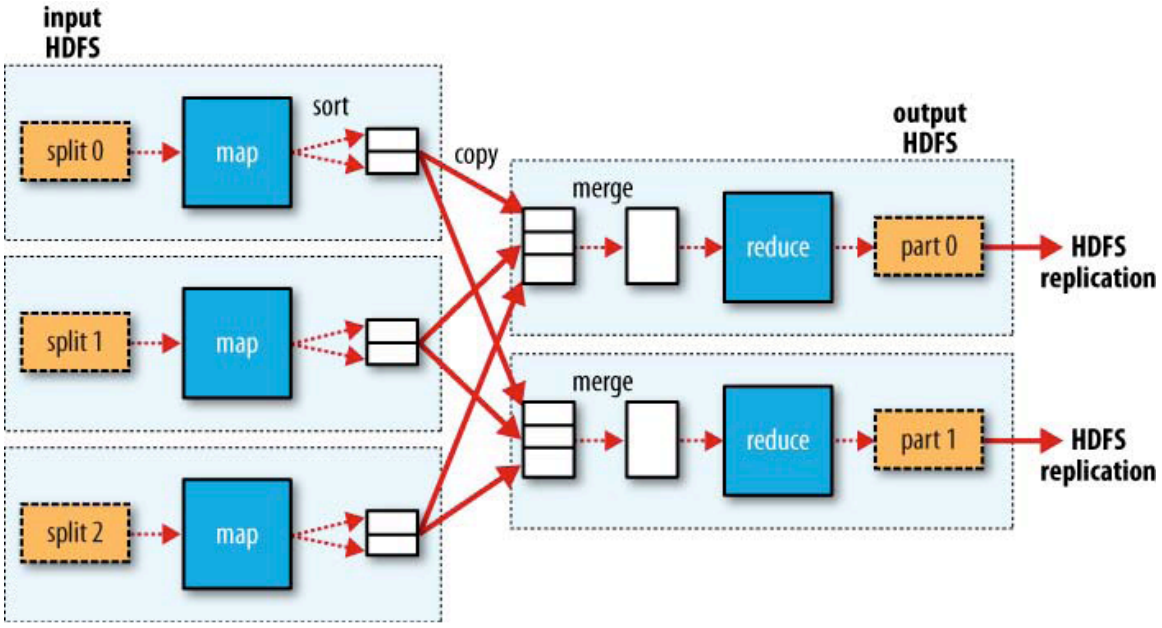


Figura 15: Diagrama de Flujo de Hadoop

2.3.7 Apache Spark

Apache Spark es un software de código abierto para computación en clúster. Proporciona una interfaz de programación de aplicaciones centrada en una estructura de datos llamada Resilient Distributed Dataset (RDD), un conjunto múltiple de elementos de datos de sólo lectura, distribuidos en un grupo de máquinas, que es tolerante a fallos.[25]

Se desarrolló en respuesta a las limitaciones en el paradigma de computación de cluster MapReduce, que obliga a una estructura de flujo de datos particularmente lineal en programas distribuidos: MapReduce lee los datos de entrada del disco duro, aplica la función Map, comparte y distribuye los resultados entre el resto de máquinas, aplica Reduce, y guarda los resultados en el disco duro. Los RDDs de Spark funcionan como un marco de trabajo para programas distribuidos que restringe deliberadamente la memoria compartida entre los nodos.

La disponibilidad de RDD facilita la implementación de algoritmos iterativos, que visitan su conjunto de datos varias veces en un bucle, y el análisis de datos interactivo o exploratorio (Figura 16). Requiere de un sistema de archivos compartido. Puede usarse NFS, HDFS, y S3, entre muchos otros. Tiene un gestor de cluster nativo, pero también se puede usar Hadoop.

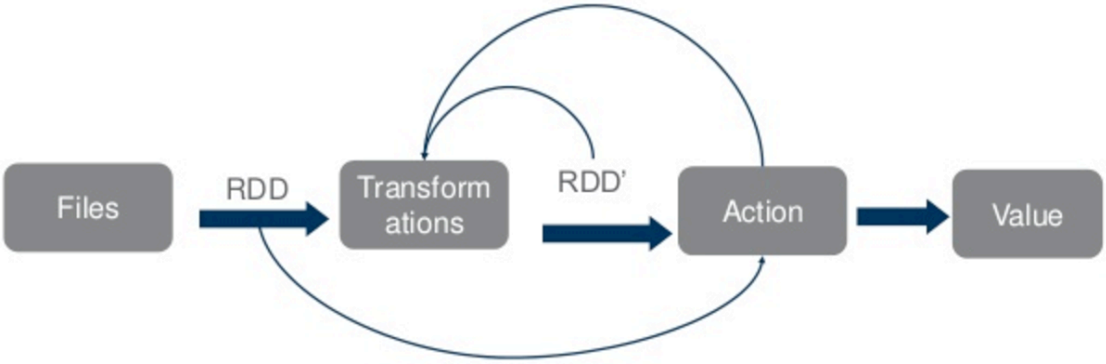


Figura 16: Diagrama de flujo de Spark

En la Figura 17 podemos ver un ejemplo de trabajo sobre RDD. Tenemos un RDD con el input inicial, *InputRDD*. Aplicamos una transformación de suma sobre cada uno de los elementos (*map*), lo que genera otro RDD (*MapRDD*). Aplicamos *count*, obteniendo el número total de elementos. Por otro lado, al RDD inicial le aplicamos un filtro, lo que produce otro RDD (*FilterRDD*), independiente de *MapRDD*. Después recopilamos los dos primeros elementos (*take*) y por otro lado guardamos todos los elementos filtrados en un archivo.

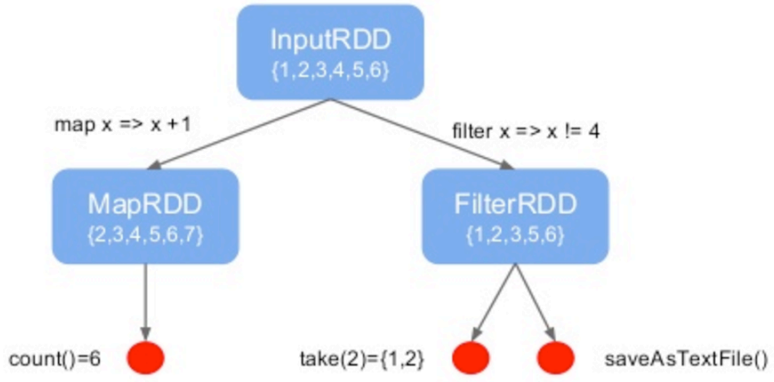


Figura 17: Ejemplo de trabajo con RDD

En Spark podemos identificar los siguientes componentes[26]:

- Maestro: Es la instancia de Spark que gestiona los esclavos en el gestor de cluster de Spark.
- Esclavo: Son las instancias de Spark que ejecutan el código de las tareas a realizar en el gestor de cluster de Spark.
- Aplicación: Es el programa usuario que se ejecutará en Spark.
- Controlador o *Driver*: El proceso que ejecuta la aplicación

- Administrador de clústeres: Un servicio para la adquisición de recursos en el clúster.

- Nodo de trabajo o *Worker*: Cualquier nodo que puede ejecutar código de la aplicación en el clúster.

- Ejecutor: Un proceso que se inicia para una aplicación, en un nodo de trabajo, que ejecuta tareas y mantiene los datos en la memoria o el almacenamiento en disco. Cada aplicación tiene sus propios ejecutores.

- Tarea: La unidad de trabajo mínima que será enviada a un ejecutor

- Trabajo: Un cálculo paralelo que consiste en varias tareas que se generan en respuesta a una acción de Spark (por ejemplo, guardar, recopilar).

- Etapa o *Stage*: Cada trabajo se divide en conjuntos más pequeños de tareas llamadas etapas que dependen entre sí (similar a MapReduce).

En la Figura 18 podemos ver que las etapas se pueden ejecutar de forma paralela e independiente en el clúster, siempre respetando las dependencias de unas y otras.

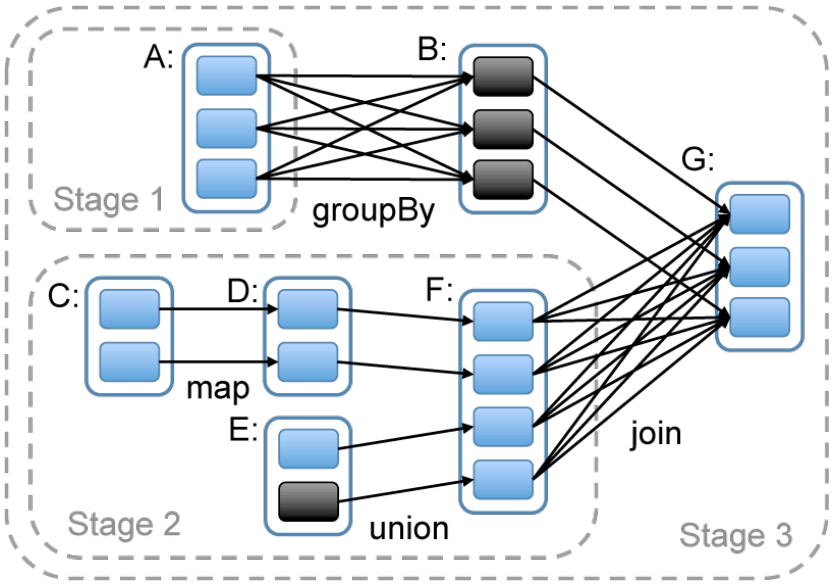


Figura 18: Ejecución de etapas

La división de tareas y etapas es organizada por el planificador DAG (*Directed Acyclic Graph*). Cuando se solicita alguna acción en el RDD, Spark crea un grafo con las operaciones y lo envía al planificador.

El planificador DAG divide los operadores en tareas y etapas. Una etapa se compone de tareas, basadas en particiones de los datos de entrada. El programador del DAG realiza tuberías (*pipes*) de operaciones, es decir, muchos operadores *map* pueden programarse en una sola etapa. El resultado final de la planificación es un conjunto de etapas.

Las etapas se pasan al programador de tareas. El programador de tareas inicia las tareas a través del gestor de clústeres. El programador de tareas no conoce las dependencias de las etapas.

Entonces los trabajadores ejecutan las tareas en el esclavo en el que está corriendo. En la Figura 19 podemos ver un diagrama con los diferentes componentes usando el gestor de clústers del propio Spark.

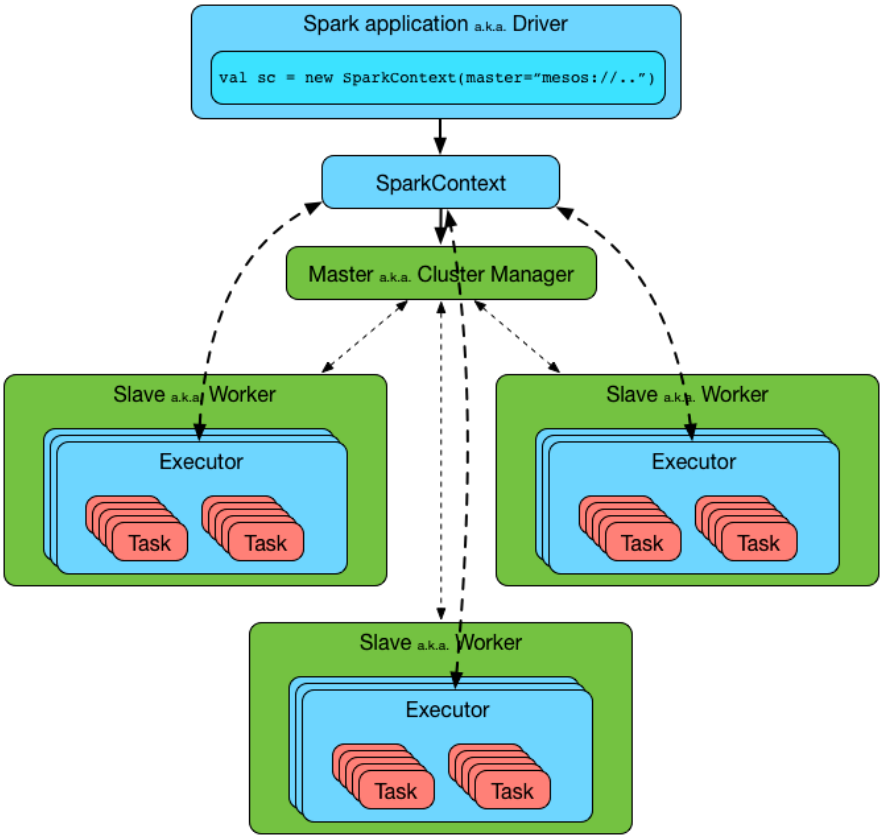


Figura 19: Planificación en Spark

Apache Hadoop ha sido ampliamente sobrepasado en popularidad por Apache Spark. En Github, podemos ver que Spark tiene hasta 4 veces más estrellas que Hadoop[27].

La librería estándar de Spark ofrece de forma nativa un módulo completo de aprendizaje automático[28]. Se puede programar en Scala, Java o Python, lo que expande aún más su entorno de librerías, especialmente con Python, que ofrece una gran cantidad de librerías de matemáticas, ciencia, e ingeniería.

En la Figura 20 podemos ver las librerías en las que se basa Apache Spark.

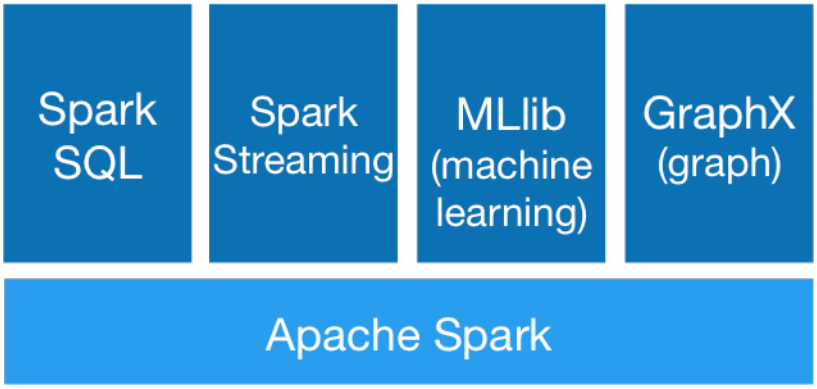


Figura 20: Librerías de Spark

2.4 Aprendizaje Automático

El aprendizaje automático o *machine learning* es un método de análisis de datos que automatiza la construcción de modelos analíticos. Usando algoritmos que iterativamente aprenden de los datos, el aprendizaje automático permite a las computadoras encontrar patrones ocultos sin ser explícitamente programado sobre dónde buscarlos[29].

Arthur Samuel acuñó el término "machine learning" en 1959 cuando trabajaba en IBM[30]. El aprendizaje de máquina explora el estudio y la construcción de algoritmos que permitan a una máquina aprender de los datos y hacer predicciones basándose en ellos, siguiendo estrictamente instrucciones estáticas.

El aprendizaje automático se emplea en una serie de tareas informáticas en las que el diseño y la programación de algoritmos explícitos con un buen rendimiento es difícil o imposible; algunos ejemplos son el filtrado de correo electrónico, la detección de intrusos de la red o ataques maliciosos, reconocimiento de caracteres ópticos, clasificación, visión por ordenador, y otros muchos.

En este caso, nos centraremos en los algoritmos de análisis de clústers o *clustering*, que es la tarea de agrupar un conjunto de objetos de tal manera que los objetos en el mismo grupo (clúster) son más similares a los demás de su propio grupo que a los de otros grupos.

Estos algoritmos son relevantes para el proyecto porque son los que nos permitirán encontrar los grupos de trazas más densos localizados geográficamente. El análisis de clústeres no es un algoritmo en sí mismo, sino que es la tarea general a resolver. Puede lograrse mediante diversos algoritmos que difieren significativamente en su noción de lo que constituye un clúster y cómo encontrarlos eficientemente. A continuación, veremos tres de ellos.

2.4.1 DBSCAN

Para clustering espacial en minería de datos, DBSCAN es uno de los algoritmos más citados[31] en las publicaciones científicas, pero no está disponible en la librería estándar de Spark.

Brevemente, DBSCAN es un algoritmo de clusterización basado en la densidad: dado un conjunto de puntos en algún espacio, agrupa puntos que están estrechamente empaquetados (puntos con muchos vecinos cercanos), marcando como puntos extremos aquellos que se encuentran solos en regiones de baja densidad (donde los vecinos están demasiado lejos).

Es especialmente eficaz para agrupamiento espacial basado en densidad con tolerancia al ruido.

En el análisis del proyecto se evaluó su uso en el sistema, pero debido a la complejidad de implementación, y que el módulo de Spark ya dispone de algunos algoritmos de análisis de clústers, se descartó a favor de estos últimos.

En la Figura 21 podemos ver un ejemplo donde cada color representa un cluster distinto, y los puntos sin color son el ruido que no ha sido incluido en ninguno de los grupos.

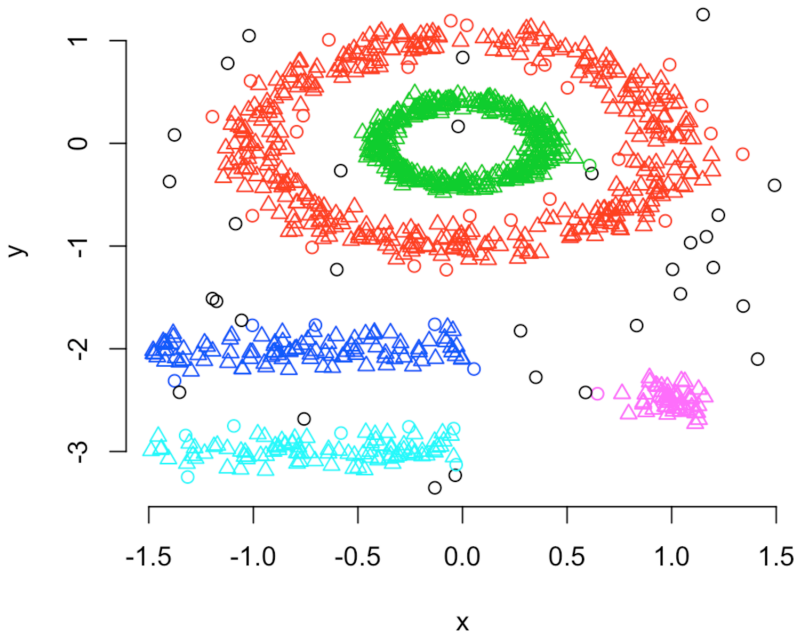


Figura 21: Ejemplo de DBSCAN

2.4.2 K-means ||

La librería de aprendizaje automático de Spark incluye el algoritmo K-means ||, el cual es una variante paralelizada de K-means ++, el cual es una variante más sofisticada

del algoritmo K-means. K-means es uno de los algoritmos de aprendizaje sin supervisión más simples de análisis de clústers[32].

El algoritmo K-means sigue una manera simple y fácil de clasificar un determinado conjunto de datos a través de un cierto número de clusters (*k*) fijados a priori. Sigue los siguientes pasos:

1. Se definen *k* centroides, uno para cada grupo. Diferentes posiciones de entrada de los centroides causan diferentes resultados. Se suelen especificar a mano o al azar, repitiendo el algoritmo buscando la configuración inicial óptima.

2. El siguiente paso es tomar cada uno de los puntos perteneciente al conjunto de datos dado y asociarlo al centroide más cercano.

3. Se recalculan *k* nuevos centroides como baricentros de los clusters resultantes y se vuelve al primer paso.

En cada iteración del bucle, los centroides cambian su ubicación paso a paso hasta que se dejan de percibir cambios.

El algoritmo tiene como objetivo minimizar una función objetivo, una función de error cuadrático. En la Figura 22 podemos ver el diagrama de flujo del algoritmo.

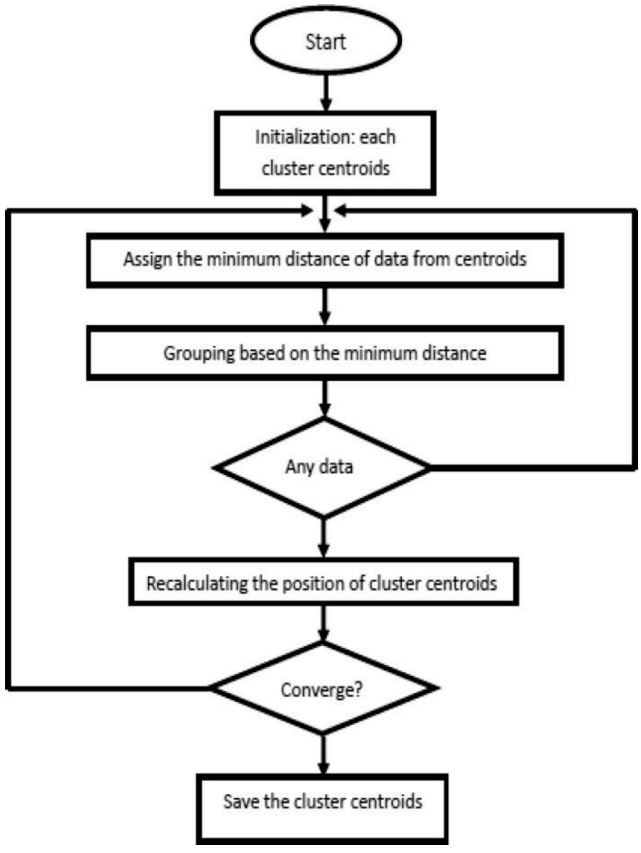


Figura 22: Diagrama de flujo K-means

En la Figura 23 podemos ver un ejemplo de clusterización con K-means. Las cruces negras representan los centroides, y los colores de cada punto el clúster al que pertenecen.

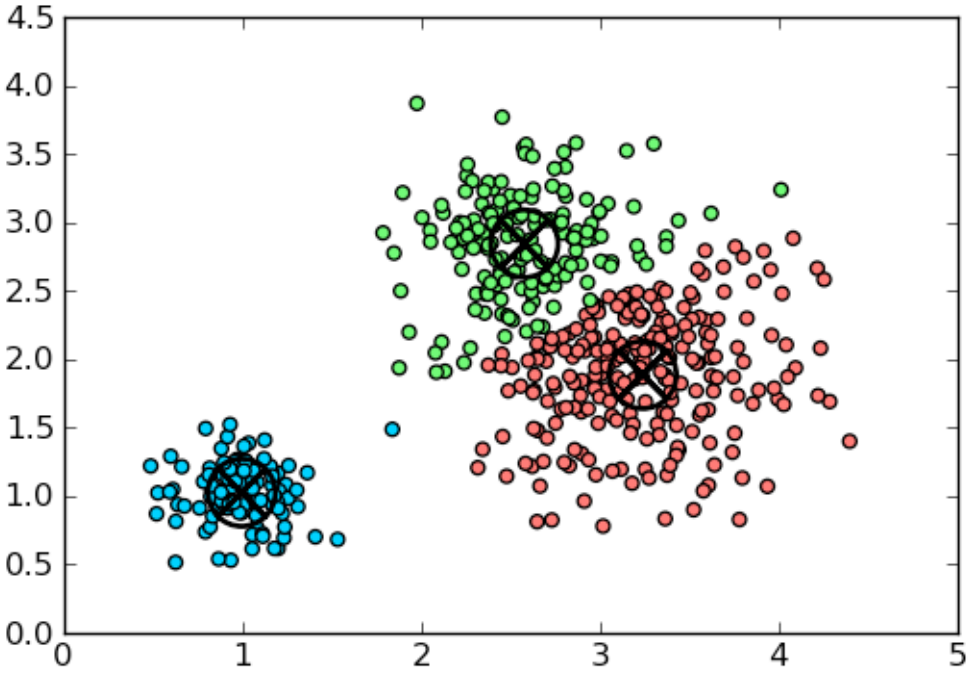


Figura 23: Ejemplo de K-means

El algoritmo K-means ++ extiende K-means tratando de encontrar la mejor inicialización de los centroides[33]. Para ello, trata de situarlos de la forma más espaciada posible, seleccionando puntos del conjunto de datos y tomándolos como centroides. Los pasos que sigue son:

1. Se elige un centro uniformemente al azar entre los puntos de datos.

2. Para cada punto de datos *x*, se calcula *D(x),* la distancia entre *x* y el centro más cercano que ya ha sido elegido.

3. Se elige un nuevo punto al azar de los datos como un nuevo centro, usando una distribución de probabilidad ponderada donde se elige un punto *x* con probabilidad proporcional a $D(x)^2$.

4. Se repiten los pasos 2 y 3 hasta que se hayan elegido los centros *k*.

5. Ahora que los centros iniciales han sido elegidos, se procede usando la agrupación estándar de k-means.

El algoritmo implementado en Spark es conocido como K-means ||. Dicho algoritmo aplica K-means ++ de forma paralelizada[34].

2.4.3 Gaussian Mixture

Hay otra manera de resolver los problemas de clustering: un enfoque basado en modelos, que consiste en usar ciertos modelos para clusters e intentar optimizar el ajuste entre los datos y el modelo.

En la práctica, cada clúster puede ser representado matemáticamente por una distribución paramétrica, como una distribución normal (Gaussiana). Por lo tanto, el conjunto de datos completo es modelado por una mezcla (*mixture*) de estas distribuciones. Una distribución individual utilizada para modelar un clúster específico se conoce a menudo como una distribución de componentes[35].

Un modelo de *mixture* suele tener los siguientes rasgos:

- Las distribuciones de los componentes tienen una gran densidad.
- El modelo *mixture* cubre ampliamente los datos (los patrones dominantes en los datos son capturados por las distribuciones de los componentes).

Las principales ventajas del clustering basado en modelos son:

- Gran disponibilidad de técnicas de inferencia estadística
- Flexibilidad en la elección de la distribución de componentes
- Obtención de estimación de densidad para cada clúster
- Se pueden realizar clasificaciones más relajadas

El método de agrupación más utilizado de este tipo es el que se basa en el aprendizaje de una mezcla de gaussianos (*Gaussian Mixture*). Dicho modelo está disponible en la librería de aprendizaje automático de Spark.

En el modelo *Gaussian Mixture*, como su nombre indica, se considera a los clusters como un conjunto de distribuciones gaussianas centradas en sus baricentros. En la Figura 24 podemos ver una mezcla de tres distribuciones gaussianas.

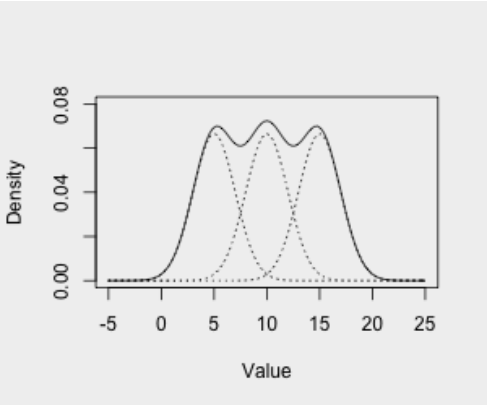


Figura 24: Mezcla de distribuciones gaussianas

En la implementación dada en Spark, el algoritmo utilizado para optimizar los clústers en cada iteración es el EM (expectativa-maximización). Es un método iterativo para encontrar estimaciones de máxima verosimilitud o máximo a posteriori (MAP) de parámetros en modelos estadísticos, donde el modelo depende de variables latentes no observadas[36]. La iteración EM alterna entre:

1. La realización del paso conocido como de expectativa (E), que crea una expectativa de la función de verosimilitud evaluada, utilizando una estimación basada en los parámetros actuales.

2. El paso de maximización (M), que calcula los parámetros maximizando la probabilidad dada en el paso E. Estas estimaciones de parámetros se utilizan para determinar la distribución de las variables latentes en el paso E siguiente.

En la figura podemos ver un ejemplo de clusterización del modelo Gaussian Mixture sobre un conjunto de datos espacial de dos dimensiones.

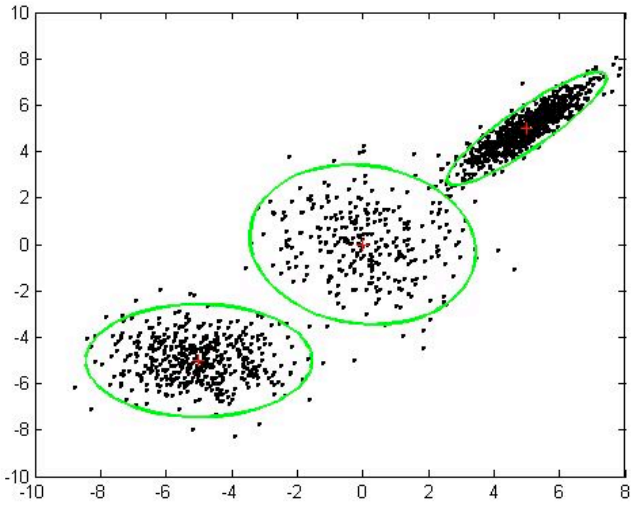


Figura 25: Ejemplo de Gaussian Mixture Model

2.5 Base de Datos

Cuando hablamos de bases de datos, nos referimos a un conjunto de datos relacionados y la forma en que se organiza.

El acceso a estos datos suele estar proporcionado por un "sistema de gestión de bases de datos" (DBMS, por sus siglas en inglés Data Base Management System) que consiste en un conjunto integrado de software que permite a los usuarios interactuar con una o más bases de datos y proporciona acceso a todos los datos contenidos en la base de datos. El DBMS proporciona varias funciones que permiten la entrada, el almacenamiento y la recuperación de grandes cantidades de información, así como diferentes formas de gestionar cómo se organiza esa información.[37] A veces se utiliza el término "base de datos" para referirse al DBMS.

Los datos suelen organizarse en una colección de esquemas, tablas, consultas, informes, vistas y otros objetos. Los diseñadores de bases de datos normalmente organizan los datos para modelar aspectos de la realidad de una manera que apoya los procesos que requieren información, como por ejemplo modelar la disponibilidad de habitaciones en hoteles de tal forma que apoyen la búsqueda de un hotel con vacantes.

Las bases de datos no son estrictamente portables a través de diferentes DBMS, pero diferentes DBMS pueden interoperar mediante el uso de estándares como SQL, ODBC o JDBC para permitir que una sola aplicación funcione con más de un DBMS. Los sistemas de gestión de bases de datos se suelen clasificar de acuerdo con los modelos de base de datos que soportan.

2.5.1 Transacciones

Una transacción simboliza una unidad de trabajo realizada dentro de una base de datos. Se realiza de una manera coherente y fiable independiente de otras transacciones. Una transacción generalmente representa cualquier cambio en una base de datos. Las transacciones en un entorno de base de datos tienen dos propósitos principales:

- Proporcionar unidades de trabajo fiables que permitan la recuperación correcta de los fallos, así como mantener una base de datos consistente incluso en casos de error del sistema, cuando la ejecución se detiene completa o parcialmente, donde las operaciones sobre una base de datos podrían quedar incompletas, con un estado incoherente.

- Proporcionar aislamiento entre los programas que acceden a una base de datos simultáneamente. Si este aislamiento no se proporciona, los resultados de los programas serán posiblemente erróneos.

Una transacción de base de datos, por definición, debe ser atómica, coherente, aislada y duradera. Dichas propiedades son conocidas con el acrónimo ACID (del inglés *Atomicity, Consistency, Isolation, Durability*), definidas por Reuter y Härder en 1983[38].

Las características de estas cuatro propiedades definidas por Reuter y Härder son las siguientes:

- Atomicidad: Cada transacción debe ejecutar "todo o nada", es decir, si una parte de la transacción falla, entonces toda la transacción falla y el estado de la base de datos no cambia. Un sistema atómico debe garantizar la atomicidad en cada situación, incluyendo fallos energéticos, errores y violación de las restricciones del esquema. Visto desde fuera, una transacción es indivisible.

- Consistencia: Garantiza que cualquier transacción llevará la base de datos de un estado válido a otro. Todos los datos escritos en la base de datos deben ser válidos de acuerdo con todas las reglas definidas, incluyendo restricciones, cascadas, disparadores y cualquier combinación de los mismos.

- Aislamiento: Asegura que la ejecución concurrente de transacciones da como resultado un estado del sistema que se obtendría si las transacciones se ejecutaban secuencialmente, es decir, una después de la otra. Proporcionar aislamiento es el objetivo principal del control de concurrencia.

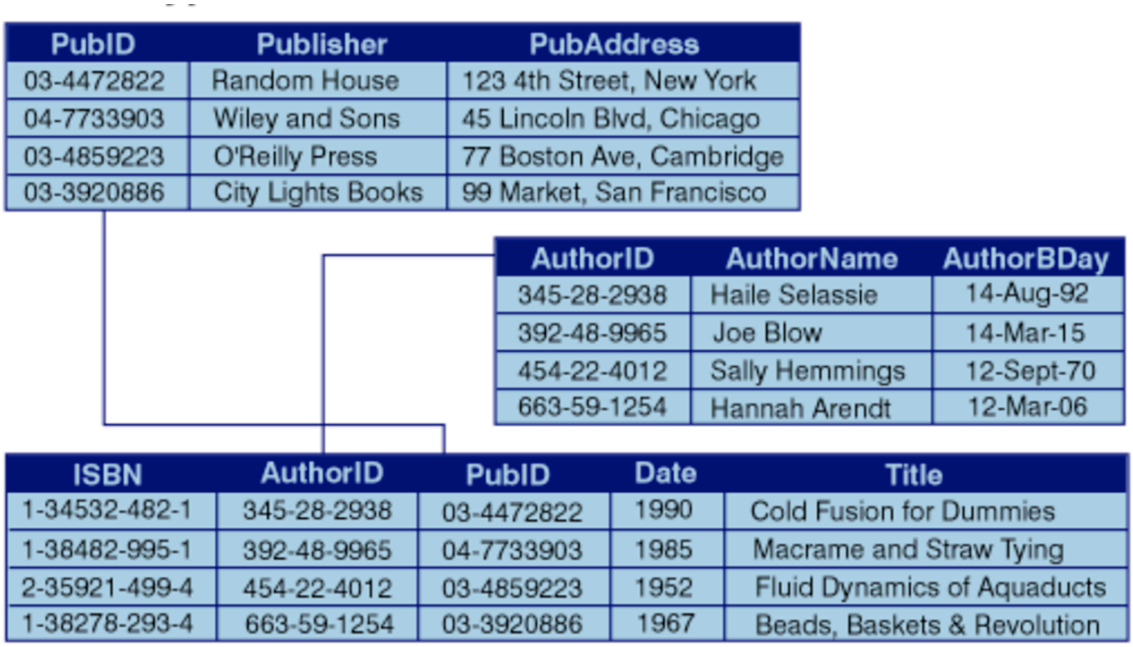
- Durabilidad: La propiedad de durabilidad asegura que una vez que una transacción se ha completado, el resultado persistirá, incluso en el caso de pérdida de energía, fallos o errores. En una base de datos relacional, por ejemplo, una vez que un grupo de instrucciones SQL se ejecuta, los resultados deben almacenarse permanentemente. Para defenderse contra la pérdida de energía, las transacciones suelen registrarse en una memoria no volátil.

2.5.2 Bases de datos relacionales

Los sistemas de bases de datos más populares desde la década de 1980 han seguido el modelo relacional, generalmente asociado con el lenguaje SQL[39].

Este modelo organiza los datos en una o más tablas de columnas y filas, conocidas como *relaciones*, con una clave única (basada en una o varias columnas) que identifica cada fila. Las tablas pueden tener columnas cuyo contenido es la clave de otra de dichas tablas, lo que permite definir relaciones entre tablas. Generalmente, cada tabla y relación representa un tipo de entidad, como por ejemplo comprador o producto. Las filas representan instancias de ese tipo de entidad (siguiendo con el ejemplo, Jorge o silla), y las columnas que representan los valores atribuidos a esa instancia (dirección o precio).

En la Figura 26 podemos ver otro ejemplo con autores y publicaciones.



PubID	Publisher	PubAddress
03-4472822	Random House	123 4th Street, New York
04-7733903	Wiley and Sons	45 Lincoln Blvd, Chicago
03-4859223	O'Reilly Press	77 Boston Ave, Cambridge
03-3920886	City Lights Books	99 Market, San Francisco

AuthorID	AuthorName	AuthorBDay
345-28-2938	Haile Selassie	14-Aug-92
392-48-9965	Joe Blow	14-Mar-15
454-22-4012	Sally Hemmings	12-Sept-70
663-59-1254	Hannah Arendt	12-Mar-06

ISBN	AuthorID	PubID	Date	Title
1-34532-482-1	345-28-2938	03-4472822	1990	Cold Fusion for Dummies
1-38482-995-1	392-48-9965	04-7733903	1985	Macrame and Straw Tying
2-35921-499-4	454-22-4012	03-4859223	1952	Fluid Dynamics of Aquaducts
1-38278-293-4	663-59-1254	03-3920886	1967	Beads, Baskets & Revolution

Figura 26: Ejemplo de base de datos relacional

SQL (Structured Query Language) es un lenguaje de programación estandarizado que se utiliza para administrar bases de datos relacionales y realizar diversas operaciones sobre los datos en ellas. Inicialmente creado en la década de 1970, SQL es utilizado regularmente por los administradores de bases de datos, así como por los desarrolladores de programas de integración con datos, y analistas de datos que buscan configurar y ejecutar consultas analíticas.

SQL permite la definición, manipulación y control de datos siguiendo el álgebra y el cálculo relacional. SQL incluye las siguientes operaciones sobre los datos:

- Inserción
- Consulta
- Actualización
- Eliminación
- Creación y modificación de esquemas
- Control de acceso a datos

Aunque SQL es un lenguaje declarativo, en algunas implementaciones también incluye elementos procedurales.

2.5.3 Bases de datos espaciales

Una base de datos espacial es una base de datos optimizada para almacenar y consultar datos que representan objetos definidos en un espacio geométrico[40].

La mayoría de las bases de datos espaciales permiten representar objetos geométricos simples como puntos, líneas y polígonos. Algunas bases de datos espaciales manejan estructuras más complejas tales como objetos 3D, coberturas topológicas, redes lineales y redes irregulares de triángulos (TIR, *Triangulated Irregular Network*).

Aunque se han desarrollado bases de datos para administrar diferentes tipos de datos, como las bases de datos relacionales, dichas bases de datos requieren funcionalidad adicional para procesar eficientemente los tipos de datos espaciales, y normalmente las bases de datos espaciales se desarrollan como una extensión de las mismas, añadiendo tipos de datos de geometría o de entidades.

Los sistemas de base de datos utilizan índices para buscar rápidamente valores, y la forma en que la mayoría de las bases de datos indexan no son óptimos para consultas espaciales. En su lugar, las bases de datos espaciales utilizan un índice espacial para acelerar las operaciones de la base de datos.

El Open Geospatial Consortium desarrolló la especificación Simple Features (publicada por primera vez en 1997) y establece estándares para añadir funcionalidades espaciales a los sistemas de bases de datos. El estándar *SQL/MM Spatial* ISO/EIC forma parte del estándar multimedia de *SQL/MM* y amplía el estándar Simple Features con tipos de datos que admiten interpolaciones circulares[41].

Además de las consultas SQL típicas, como las sentencias SELECT, las bases de datos espaciales pueden realizar una amplia variedad de operaciones espaciales. Las siguientes operaciones (entre otras) son especificadas por el estándar Open Geospatial Consortium:

- Medidas espaciales: Calcula la longitud de la línea, área del polígono, distancia entre geometrías, etc.

- Funciones espaciales: Amplia las funciones SQL existentes para crear otras nuevas, como por ejemplo obtener un círculo alrededor de un punto dado, funciones de intersección, etc.

- Predicados Espaciales: Permite consultas con resultados booleanos (verdadero/falso) sobre relaciones espaciales entre geometrías, como por ejemplo superposición de polígonos o presencia de entidades en una distancia determinada.

- Constructores de geometría: Crea nuevas geometrías, usualmente especificando los vértices (puntos o nodos) que definen la forma.

- Funciones de observación: Consultas que devuelven información específica sobre una característica, como la ubicación del centro de un círculo.

2.5.4 MongoDB

Se incluye brevemente dado que se evaluó para su uso en el proyecto.

Es una base de datos orientada a documentos. Describe los documentos en formato JSON, y al contrario que una base de datos relacional, no necesita un esquema, aunque también es posible. Es una de las bases de datos orientada a documentos más populares. Existen interfaces para los lenguajes más populares.

Esta base de datos incluye consultas espaciales de forma nativa[42].

2.5.5 PostgreSQL y PostGIS

PostgreSQL es una base de datos relaccional, ACID y transaccional. Nacida en 1986 en la Universidad de Berkeley, es una de las bases de datos relaccionales más populares[43].

PostGIS es una extensión para PostgreSQL[44]. Añade soporte para objetos geográficos permitiendo que las consultas de ubicación se ejecuten en SQL. El lenguaje SQL tiene interfaces para los lenguajes más populares.

PostGIS sigue las "Simple Features for SQL Specification" del Open Geospatial Consortium y ha sido certificado como compatible con el perfil "Types and Functions". PostGIS es software libre, publicado bajo la GNU General Public License[45].

En la Figura 27 podemos ver una consulta de ejemplo en PostGIS. En ella, se piden todos los registros cuya distancia sea menor de 100 unidades respecto de un punto concreto.

```
SELECT the_geom

FROM geom_table

WHERE ST_Distance(the_geom, ST_GeomFromText('POINT(100000 200000)')) < 100
```

Figura 27: Ejemplo de consulta PostGIS

2.6 Aplicaciones Web

Una aplicación web es un programa cliente-servidor en el que el cliente es ejecutado en un navegador web[46], el cual es a su vez otro programa que permite al usuario navegar por internet. Un navegador web hace las consultas a los servidores web, recibe los documentos a mostrar, los pinta, y permite al usuario interactuar con ellos. Los documentos están construidos sobre los lenguajes HTML, CSS y JavaScript.

Por tanto, todas las aplicaciones web están también construidas sobre HTML, CSS, HTML, y Javascript. Los documentos se consultan, modifican y transmiten por el protocolo conocido como HTTP. Dicho protocolo se situaría en la capa 7 del modelo OSI (Aplicación)[47].

El modelo de OSI (*Open Systems Interconnection*) es un modelo conceptual que caracteriza y estandariza las funciones de comunicación de un sistema de telecomunicaciones o de computación sin tener en cuenta su estructura interna y su tecnología subyacentes[48]. Su objetivo es la interoperabilidad de diversos sistemas de comunicación con protocolos estándar. El modelo divide un sistema de comunicación en siete capas de abstracción. En la Figura 28 podemos ver las capas del modelo.

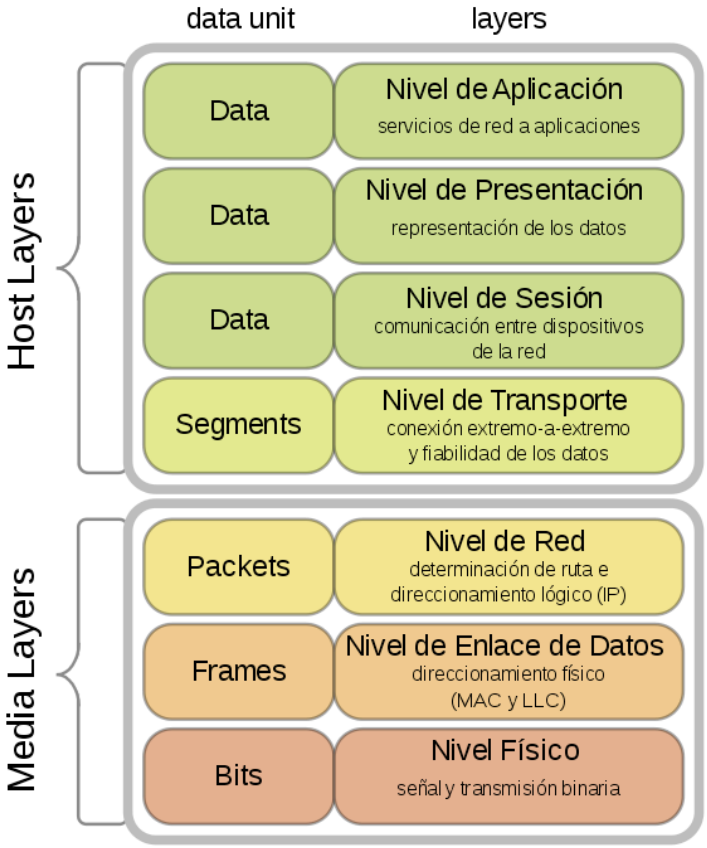


Figura 28: Capas del modelo OSI [49]

Hay multitud de marcos de trabajo que facilitan el desarrollo de aplicaciones web. Se suelen clasificar principalmente dos tipos, *Frontend* y *Backend*[50], los cuales describiremos más adelante.

2.6.1 Modelo – Vista – Controlador

El modelo MVC (Modelo-Vista-Controlador) es un patrón de diseño de software para implementar interfaces de usuario en equipos. Tradicionalmente utilizada para interfaces gráficas de usuario, este patrón se ha vuelto especialmente popular para diseñar aplicaciones web e incluso clientes móviles, aplicaciones de escritorio, y otros.

El patrón propone dividir la aplicación en tres componentes interconectados. Esto se hace para separar las representaciones internas de la información de las formas en que se presenta la información y las interacciones del usuario. El patrón de diseño MVC desacopla estos componentes principales permitiendo la reutilización eficiente del código y el desarrollo en paralelo.

La comunicación entre los componentes puede variar según la implementación. En el caso de las aplicaciones web, en la Figura 29 podemos ver una de las implementaciones más populares.

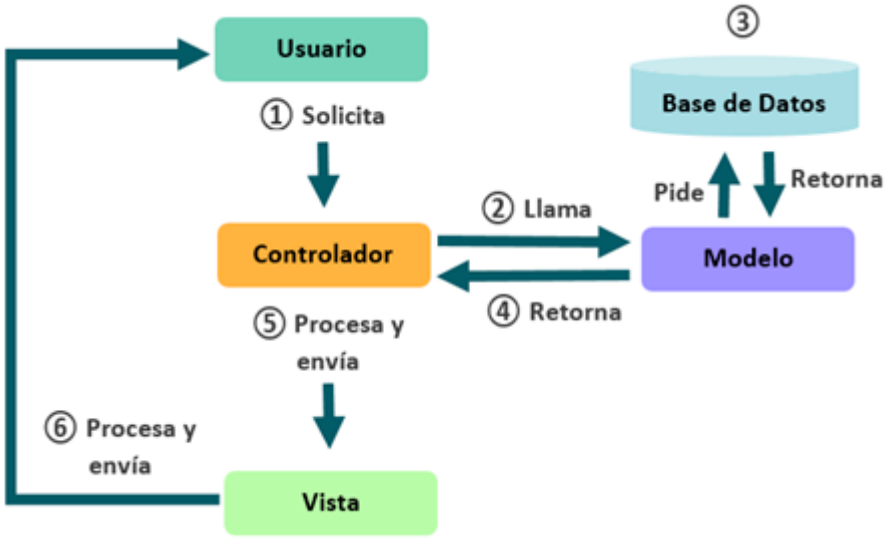


Figura 29: Modelo - Vista - Controlador

2.6.2 Frontend

El frontend es la capa de presentación. Consiste en el pintado del documento HTML, aplicando los estilos CSS, y la dinaminación de la página en JavaScript.

Existen marcos de trabajo de código abierto para crear frontends muy sofisticados, pero los requisitos del frontend en este proyecto son muy sencillos y no serán necesarios.

Es importante señalar que la integración que incluiremos con Google Maps se realiza en el frontend. Google Maps ofrece una API[51] para la integración de páginas web externas junto con su mapa. Entre otras características, permite programar la interacción con el mapa, así como su visualización. En la Figura 30 podemos ver un ejemplo de la documentación oficial de Google Maps en el que vemos la posición de terremotos.

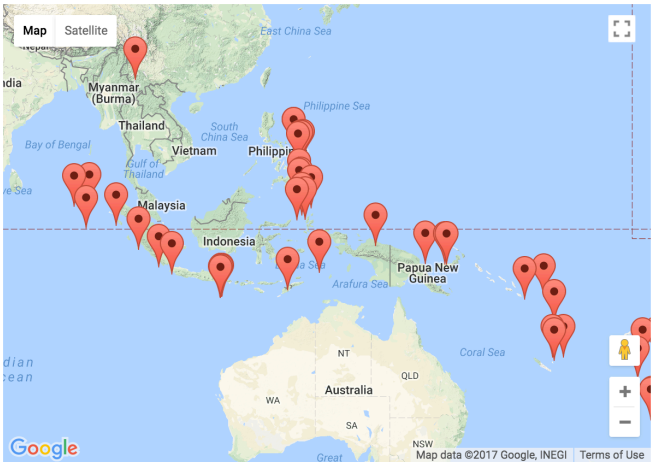


Figura 30: Ejemplo de integración con Google Maps API

2.6.3 Backend

El backend se encarga de acceder a los datos y enviar los documentos necesarios al cliente para que los visualice. Suelen seguir una arquitectura MVC[52]. Hay marcos de trabajo para prácticamente todos los lenguajes. Algunos ejemplos son Django para Python, RAILS para Ruby, y Symfony para PHP, entre otros.

2.6.4 Django

Django es un marco de trabajo (*framework*) para el desarrollo del backend de aplicaciones web. Está desarrollado en Python y sigue una variación del patrón MVC conocida como MVT (*Model-View-Template,* Modelo-Vista-Plantilla). En el fondo es el mismo patrón de software, pero los nombres son distintos:

- El Controlador pasa a llamarse Vista
- La Vista pasa a llamarse Plantilla (*Template*)

El principal objetivo de Django es facilitar la creación de sitios web complejos basados en bases de datos. Django hace hincapié en la reutilización y la conectividad de los componentes, el desarrollo ágil, y el principio de no repetirse (*DRY, Don't Repeat Yourself*). Todo el sistema está basado en Python, incluso para archivos de configuración y modelos de datos. Django también proporciona una interfaz de administración opcional que permite la creación, lectura, actualización y eliminación. Dicha interfaz es generada dinámicamente a través de la introspección y configurada a través de modelos de administración.

Hay dos características de Django que la convierten en un gran candidato para este proyecto:

- Interfaz para PostgreSQL
- Interfaz para PostGIS

- Interfaz gráfica de administración con edición de formas geométricas sobre mapas basado en PostGIS. Para ver capturas de la interfaz, por favor consultar el Anexo A.

2.7 Internet Hosting

Un servicio de alojamiento de Internet (*Internet Hosting*) es un servicio que ejecuta servidores de Internet, lo que permite a las organizaciones y a los individuos servir contenido a Internet. Hay varios niveles de servicio y varios tipos de servicios ofrecidos.

La minería de datos la realizaremos una única vez en el clúster universtario, pero posteriormente tendremos que alojar la web en un servidor en internet que esté disponible las 24 horas los 7 días de la semana para que los taxistas puedan consultarla en cualquier momento.

2.7.1 Digital Ocean

DigitalOcean, Inc. es un proveedor estadounidense de servidores en internet con sede en la ciudad de Nueva York. Tienen centros de datos en todo el mundo. DigitalOcean también proporciona a los desarrolladores servicios en la nube que ayudan a implementar y escalar aplicaciones que se ejecutan simultáneamente en varios equipos.

La motivación principal por la que se escogió dicho proveedor, es que los precios son muy competitivos. En diciembre de 2015, DigitalOcean fue la segunda empresa de hosting más grande del mundo en términos de equipos orientados a la web[53].

Ofrecen servidores privados virtuales (*VPS, Virtual Private Server*), empezando con 5$ al mes. Un VPS ejecuta su propia copia de un sistema operativo (SO), y los clientes pueden tener acceso a nivel superusuario a esa instancia del sistema operativo, por lo que pueden instalar casi cualquier software que se ejecuta en él. En general, son funcionalmente equivalentes a un servidor físico dedicado, y al estar definidos por software, pueden ser creados y configurados mucho más fácilmente. Su precio es mucho menor que un servidor físico equivalente. Sin embargo, como comparten el hardware físico subyacente con otros VPS, el rendimiento puede ser menor, dependiendo de la carga de trabajo de las otras máquinas virtuales en ejecución.

2.7.2 Debian

Debian es uno de los sistemas operativos que Digital Ocean ofrece como instalación base para las máquinas virtuales, y es también el sistema operativo utilizado por los ordenadores en el clúster universitario.

Debian es una de las distribuciones del sistema operativo Linux más popular para computadoras personales y servidores de red, y se ha utilizado como base para muchas otras distribuciones.

Debian es un sistema operativo de tipo Unix que se compone enteramente de software libre, la mayoría de los cuales están bajo la GNU General Public License. Se empaquetan por un grupo de individuos que participan en el Proyecto Debian.

Capítulo 3: Análisis

Aquí realizaremos un estudio de los requisitos que deberá cumplir el sistema big data.

Por un lado, definiremos los requisitos del proyecto, tanto de la computación de las recomendaciones en el clúster como de su presentación posterior en forma de aplicación web.

Por otro lado, primero exploraremos el dataset, veremos qué información podemos extraer, y buscaremos conocimiento que podamos aplicar posteriormente en el diseño del modelo del recomendador para taxistas.

3.1 Requisitos de usuario

Debido a la falta de un cliente, los requisitos han sido establecido por el autor y el tutor.

Para la extracción de conocimiento, hemos decidido utilizar Apache Spark principalmente por cuatro razones:

- Contiene librerías de matemáticas y machine learning que podremos aplicar en el proceso.
- Como hemos visto en el Estado del Arte, es especialmente útil para exploraciones interactivas de los datos.
- Parte de los algoritmos explorados en el análisis interactivo de los datos que hagamos aquí lo podremos aplicar posteriormente en la aplicación que implemente el modelo a computar en el clúster.
- Ofrece interfaz en Python, permitiendo usar librerías de matemáticas e ingeniería con la que el autor tiene algo de experiencia previa: la librería SciPy[54].

Para la persistencia de las recomendaciones, las consultas geoespaciales, y la aplicación web, hemos decidido usar PostgreSQL con la extensión PostGIS, junto con el *framework* web Django dado que:

- Django ofrece una interfaz programática con PostgreSQL y PostGIS.
- Django ofrece una interfaz de administración para PostGIS especializada en la gestión de polígonos y mapas. Véase el Anexo A para ejemplos de capturas.

Cada requisito de usuario estará definido por los siguientes campos:

- **Identificador:** Es el identificador del requisito. Seguirá el formato RU-(C|R)-XX, en el cual:
    - **RU**: Establece al requisito como de usuario (Requisito de Usuario).
    - **C** o **R**: Establece si el requisito es de capacidad (C) o de restricción (R).
        - Capacidad: Especifican la funcionalidad o servicios que la aplicación debe proporcionar.
        - Restricción: Imponen restricciones en el producto desarrollado y en el proceso de desarrollo.
    - **XX**: Un identificador numérico del requisito que comprende entre 00 a 99. La secuencia es independiente de cada tipo de requisito (de usuario o de software).
- **Fuente**: El autor o el tutor del proyecto
- **Necesidad**: Importancia del requisito
    - Fundamental: Es un requisito obligatorio
    - Recomendable: Es un requisito deseable
    - Prescindible: Es un requisito opcional
- **Prioridad**: Orden de importancia para la planificación. Podrá ser *alta*, *media* o *baja*.
- **Descripción**: Explicación del requisito

A continuación, se exponen los requisitos

Identificador	RU-C-01
Fuente	Tutor
Necesidad	Fundamental
Prioridad	Alta
Descripción	
El dataset a utilizar será el publicado por el NYC Taxi & Limousine Commission.	

Tabla 1: RU-C-01

Identificador	RU-C-02
Fuente	Tutor
Necesidad	Fundamental
Prioridad	Alta
Descripción	
El sistema debe ser capaz de procesar un gran volumen de datos.	

Tabla 2: RU-C-02

Identificador	RU-C-03
Fuente	Tutor
Necesidad	Fundamental
Prioridad	Alta
Descripción	
El sistema debe ser capaz de eliminar los registros inválidos	

Tabla 3: RU-C-03

Identificador	RU-C-04
Fuente	Tutor
Necesidad	Fundamental
Prioridad	Alta
Descripción	
El sistema debe ser capaz de extraer recomendaciones para taxistas de los datos.	

Tabla 4: RU-C-04

Identificador	RU-C-05
Fuente	Tutor
Necesidad	Fundamental
Prioridad	Alta
Descripción	
El sistema debe poder ejecutarse en entornos de diferentes niveles de potencia de cómputo.	

Tabla 5: RU-C-05

Identificador	RU-C-06
Fuente	Tutor
Necesidad	Fundamental
Prioridad	Alta
Descripción	
Se debe comparar el rendimiento a diferentes niveles de potencia de cómputo.	

Tabla 6: RU-C-06

Identificador	RU-C-07
Fuente	Tutor
Necesidad	Fundamental
Prioridad	Alta
Descripción	
Se debe comparar el rendimiento del sistema con diferentes tamaños de entrada.	

Tabla 7: RU-C-07

Identificador	RU-C-08
Fuente	Autor
Necesidad	Recomendable
Prioridad	Baja
Descripción	
El sistema ofrecerá una interfaz de usuario para consultar las recomendaciones por los taxistas.	

Tabla 8: RU-C-08

Identificador	RU-C-09
Fuente	Autor
Necesidad	Prescindible
Prioridad	Baja
Descripción	
El sistema ofrecerá una interfaz de usuario para administrar las recomendaciones por un administrador.	

Tabla 9: RU-C-09

Identificador	RU-C-10
Fuente	Tutor
Necesidad	Fundamental
Prioridad	Alta
Descripción	
El sistema ofrecerá una interfaz de usuario por línea de comandos de Linux para computar las recomendaciones.	

Tabla 10: RU-C-10

Identificador	RU-R-01
Fuente	Tutor
Necesidad	Fundamental
Prioridad	Alta
Descripción	
El sistema deberá usar Apache Spark para realizar la computación de las recomendaciones.	

Tabla 11: RU-R-01

Identificador	RU-R-02
Fuente	Autor
Necesidad	Recomendable
Prioridad	Alta
Descripción	
El sistema deberá implementarse en Python	

Tabla 12: RU-R-02

Identificador	RU-R-03
Fuente	Autor
Necesidad	Recomendable
Prioridad	Media
Descripción	
El sistema deberá usar PostGIS para almacenar y consultar las recomendaciones.	

Tabla 13: RU-R-03

Identificador	RU-R-04
Fuente	Autor
Necesidad	Recomendable
Prioridad	Baja
Descripción	
El sistema deberá usar Django para realizar la aplicación web.	

Tabla 14: RU-R-04

Identificador	RU-R-05
Fuente	Autor
Necesidad	Fundamental
Prioridad	Alta
Descripción	
Se deben documentar análisis, diseño, implementación y pruebas del sistema.	

Tabla 15: RU-R-05

3.2 Requisitos de software

Se procede a definir los requisitos de software, teniendo en cuenta los requisitos de usuario vistos anteriormente.

El formato utilizado es el siguiente:

- **Identificador:** Es el identificador del requisito. Seguirá el formato RU-(F|NFR|NFD)-XX, en el cual:
  - **RS**: Establece al requisito como de software (Requisito de Software).
  - **F**, **NF-R** o **NF-D**: Establece si el requisito es de funcional (F), no funcional de documentación (NFR), o no funcional de restricción (NFD).
    - Documentación: Especifican la documentación que necesitará.
    - Restricción: Imponen restricciones en el desarrollo.
  - **XX**: Un identificador numérico del requisito que comprende entre 00 a 99. La secuencia es independiente de cada tipo de requisito (de usuario o de software).
- **Fuente**: El requisito de usuario que enfrenta
- **Necesidad**: Importancia del requisito, igual que los requisitos de usuario.
- **Prioridad**: Orden de importancia para la planificación, igual que los requisitos de usuario.
- **Descripción**: Explicación del requisito

A continuación, se exponen los requisitos.

Identificador	RS-F-01
Fuente	RU-C-01
Necesidad	Fundamental
Prioridad	Alta
Descripción	
Se usará el fichero de Enero de 2016 del dataset requerido. En formato CSV, la primera línea es la cabecera, el resto de líneas son trazas. Se da una línea por traza, contiene 10906859. Pesa 1.6 GB.	

Tabla 16: RS-F-01

Identificador	RS-F-02
Fuente	RU-C-02
Necesidad	Fundamental
Prioridad	Alta
Descripción	
El sistema debe ser capaz de procesar todas las trazas sin errores de ejecución.	

Tabla 17: RS-F-02

Identificador	RS-F-03
Fuente	RU-C-03
Necesidad	Fundamental
Prioridad	Alta
Descripción	
El sistema debe ser capaz de detectar registros con datos inválidos e ignorarlos.	

Tabla 18: RS-F-03

Identificador	RS-F-04
Fuente	RU-C-04
Necesidad	Fundamental
Prioridad	Alta
Descripción	
El sistema debe ser capaz de recomendar las localizaciones más lucrativas para el taxista.	

Tabla 19: RS-F-04

Identificador	RS-F-05
Fuente	RU-C-04
Necesidad	Fundamental
Prioridad	Alta
Descripción	
El sistema debe ser capaz de filtrar las recomendaciones por hora y día de la semana.	

Tabla 20: RS-F-05

Identificador	RS-F-06
Fuente	RU-C-05
Necesidad	Fundamental
Prioridad	Alta
Descripción	
El sistema podrá ejecutarse sobre entre 1 y 5 esclavos, y cada uno de ellos podrá tener entre 1 y 2 trabajadores. Se ejecutará en el clúster universitario.	

Tabla 21: RS-F-06

Identificador	RS-F-07
Fuente	RU-C-06
Necesidad	Fundamental
Prioridad	Alta
Descripción	
El sistema registrará los tiempos de ejecución resultantes.	

Tabla 22: RS-F-07

Identificador	RS-F-08
Fuente	RU-C-07
Necesidad	Fundamental
Prioridad	Alta
Descripción	
El sistema podrá ejecutarse con un número de trazas y un número diferente de localizaciones a calcular.	

Tabla 23: RS-F-08

Identificador	RS-F-09
Fuente	RU-C-08
Necesidad	Recomendable
Prioridad	Baja
Descripción	
La aplicación web ofrecerá una página con un formulario para hacer las consultas de las recomendaciones.	

Tabla 24: RS-F-09

Identificador	RS-F-10
Fuente	RU-C-09
Necesidad	Prescindible
Prioridad	Baja
Descripción	
La aplicación web ofrecerá una página de administración con un listado y un formulario para gestionar las recomendaciones.	

Tabla 25: RS-F-10

Identificador	RS-F-11
Fuente	RU-C-10
Necesidad	Fundamental
Prioridad	Alta
Descripción	
La ejecución de la computación de las recomendaciones se realizará a través de la línea de comandos de Linux.	

Tabla 26: RS-F-11

Identificador	RS-NF-R-01
Fuente	RU-R-01
Necesidad	Fundamental
Prioridad	Alta
Descripción	
Se usará Apache Spark para realizar la computación de las recomendaciones.	

Tabla 27: RS-NF-R-01

Identificador	RS-NF-R-02
Fuente	RU-R-02
Necesidad	Recomendable
Prioridad	Alta
Descripción	
El sistema se implementará en Python	

Tabla 28: RS-NF-R-02

Identificador	RS-NF-R-03
Fuente	RU-R-03
Necesidad	Recomendable
Prioridad	Media
Descripción	
La base de datos se realizará en PostgreSQL y PostGIS	

Tabla 29: RS-NF-R-03

Identificador	RS-NF-R-04
Fuente	RU-R-04
Necesidad	Recomendable
Prioridad	Baja
Descripción	
La aplicación web se realizará en Django.	

Tabla 30: RS-NF-R-04

Identificador	RS-NF-D-01
Fuente	Autor
Necesidad	Fundamental
Prioridad	Alta
Descripción	
Se escribirá una memoria documentando todo el proceso.	

Tabla 31: RS-NF-D-01

3.3 Matriz de trazabilidad

	RU-C-01	RU-C-02	RU-C-03	RU-C-04	RU-C-05	RU-C-06	RU-C-07	RU-C-08	RU-C-09	RU-C-10	RU-R-01	RU-R-02	RU-R-03	RU-R-04	RU-R-05
RS-F-01	X														
RS-F-02		X													
RS-F-03			X												
RS-F-04				X											
RS-F-05				X											
RS-F-06					X										
RS-F-07						X									
RS-F-08							X								
RS-F-09								X							
RS-F-10									X						
RS-F-11										X					
RS-NF-R-01											X				
RS-NF-R-02												X			
RS-NF-R-03													X		
RS-NF-R-04														X	
RS-NF-D-01															X

Capítulo 4: Diseño

4.1 Introducción

Una vez definidos los requisitos, diseñaremos el sistema. En este capítulo abordaremos los pasos que hemos seguido en el diseño que dará solución a los objetivos del proyecto.

El sistema estará compuesto por tres componentes principales:

- El modelo de recomendador, que calculará las recomendaciones. Dicho modelo se implementará como una aplicación para Spark.
- La base de datos espacial, que persitirá las recomendaciones y permitirá realizar consultas.
- La aplicación web, que interpretará los resultados y permitirá a los taxistas interactuar con ellos.

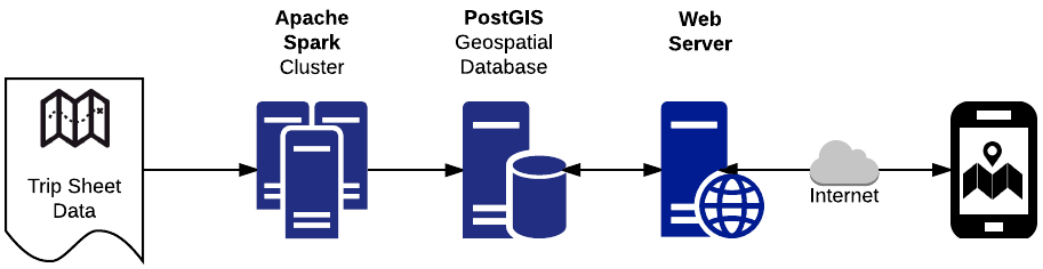


Figura 31. Visión general del sistema

4.2 Arquitectura

En esta sección, diseñaremos la arquitectura necesaria por los componentes del sistema.

4.2.1 Modelo recomendador

El modelo recomendador tiene como tarea computar las recomendaciones mediante minería de datos y aprendizaje automático, utilizando un clúster Apache Spark.

Como vimos en el Estado del Arte, Apache Spark tiene esencialmente dos requisitos:

- Necesita un gestor de recursos de clúster
- Necesita un sistema de ficheros distribuido

Como gestor de recursos de clúster, Apache Spark incluye un gestor autónomo, conocido como *Standalone mode*[55]. Es el modo más sencillo de uso, y requiere una mínima configuración. Como en otras arquitecturas de computación en clúster, el gestor organiza el clúster en maestros y esclavos.

Gracias al Departamento de Ingeniería Telemática de la universidad, tenemos disponible el clúster universitario, con seis máquinas de alta potencia, con cuatro cores y 16 GB de RAM. Podremos desplegar aquí el clúster, teniendo hasta un maestro y cinco esclavos, como muestra la Figura 38.

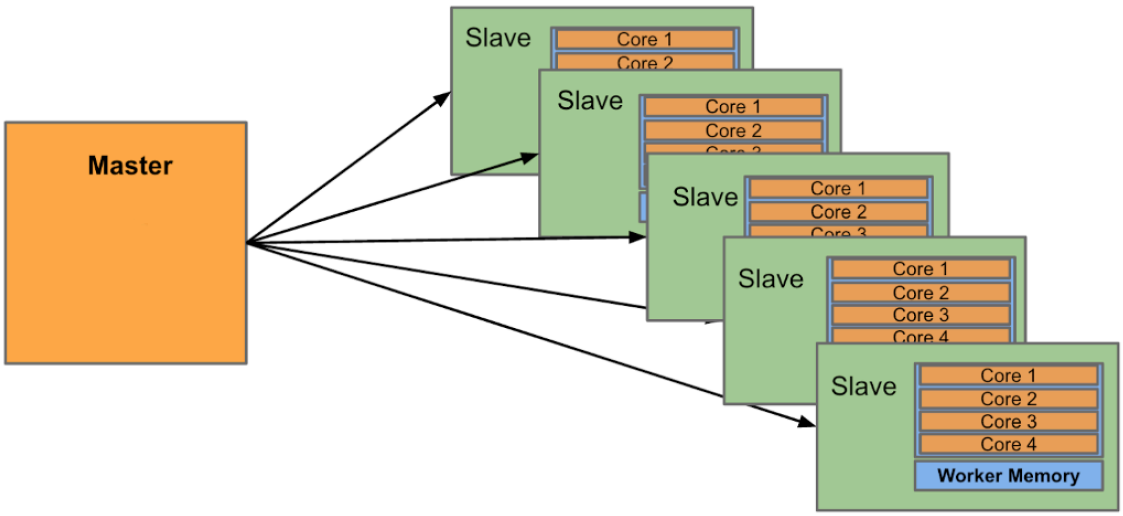


Figura 32: Apache Spark en modo *Standalone*

Como sistema de ficheros distribuido, podemos aprovecharnos de la arquitectura del clúster universitario, que utiliza NFS (*Network File System*), un sistema de archivos distribuido diseñado por Sun Microsystems[56]. El sistema permite montar volúmenes remotos como si se encontraran en la máquina local. En el caso del clúster universitario, el sistema NFS está basado en un servidor Debian GNU/Linux ubicado en la misma red.

4.2.2 Base de datos y Aplicación Web

La aplicación web cubre un requisito opcional, que es poner a disposición de los taxistas una página web online donde puedan consultar las recomendaciones. No se han requerido pruebas de carga sobre el servicio.

No esperamos un tráfico muy alto, por lo que consideramos razonable desplegar todos los componentes en un solo VPS de características mínimas. Escogemos la opción más básica en DigitalOcean[57]: 1 core, 500MB de RAM, y 20 GB de disco duro.

Por tanto, la arquitectura de la aplicación web constará tan sólo de un servidor web, con una IP pública que permita su acceso por internet.

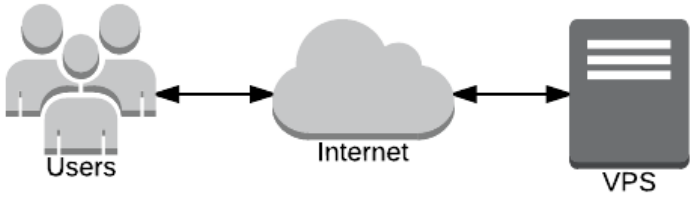


Figura 33: Arquitectura de la base de datos y aplicación web

4.3 Componentes

A continuación, diseñaremos los detalles de los componentes del sistema para los diferentes escenarios.

4.3.1 Escenario 1: Modelo recomendador

El propósito de este escenario es leer el dataset, computar las recomendaciones, y retornarlas al usuario. El sistema tendrá un único usuario, el administrador del modelo. El sistema ofrecerá una interfaz tal que el administrador pueda:

- Ejecutar el cómputo de recomendaciones.
- Obtener los resultados de la computación, para posteriormente volcarlos en la base de datos espacial.

En la Figura 34 podemos identificar tres componentes:

- Carga de datos: Es el primer componente en ser ejecutado. Su tarea consiste en cargar el dataset y prepararlo para ser procesado.
- Procesamiento de datos: Consiste en la computación de las recomendaciones con aprendizaje automático.
- Persistencia de resultados: Las recomendaciones obtenidas deben guardarse en memoria para su posterior uso por parte del usuario.

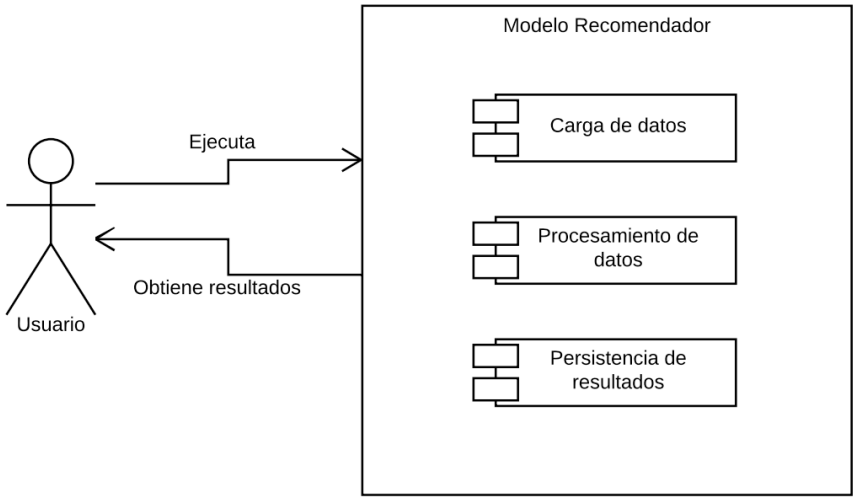


Figura 34: Componentes del Modelo Recomendador

4.3.2 Escenario 2: Transferencia a la base de datos

El propósito de este escenario es cargar los resultados en la base de datos espacial. El sistema tendrá un único usuario, el administrador de la base de datos. El sistema ofrecerá una interfaz tal que el administrador pueda:

- Ejecutar el volcado a la base de datos de los resultados obtenidos en el Escenario 1.

En la Figura 35 podemos identificar dos componentes:

- Carga de datos: El componente lee los resultados obtenidos en el Escenario 1 y valida que el formato es correcto.

- Persistencia de resultados: Las recomendaciones se guardan en la base de datos espacial.

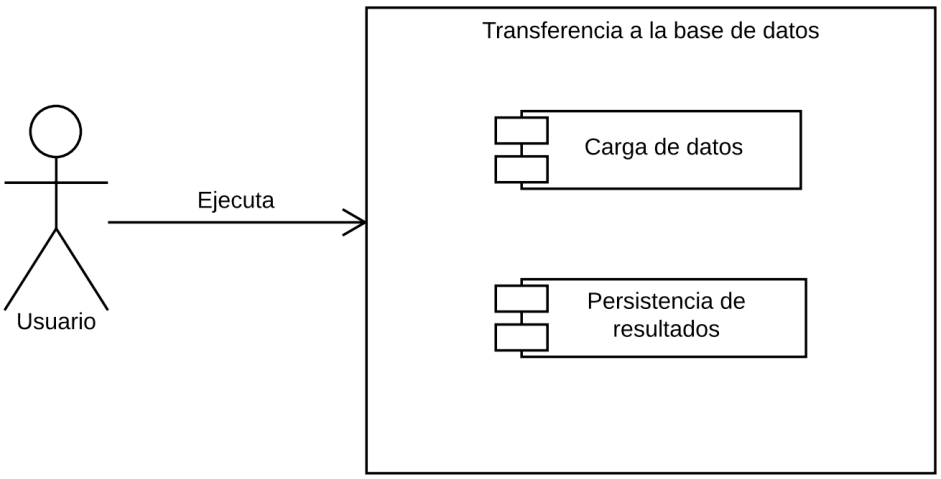


Figura 35: Componentes de la Transferencia a la base de datos

4.3.3 Escenario 3: Consulta de los resultados

El propósito de este escenario es permitir a los taxistas consultar las recomendaciones más relevantes en un momento y ubicación dados. El sistema tendrá un único usuario, el conductor del taxi. El sistema ofrecerá una interfaz tal que el usuario pueda:

- Consultar las recomendaciones más relevantes.

En la podemos identificar dos componentes:

- Servidor de consultas: El componente interpreta la consulta del usuario, y transfiere la petición al Almacén de datos.
- Almacén de datos: Contiene las recomendaciones computadas en el Escenario 1.

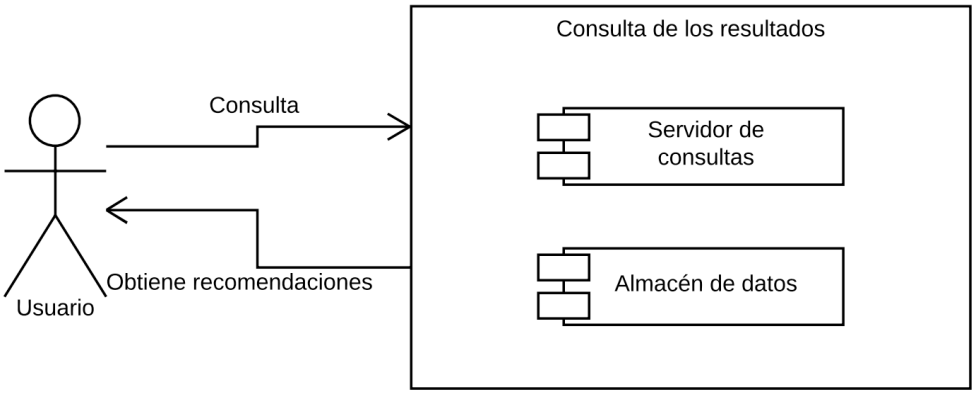


Figura 36: Componentes de la Consulta de los resultados

Capítulo 5: Implementación

5.1 Introducción

Una vez diseñado el sistema, pasamos a su implementación. En este capítulo abordaremos en la implementación que dará solución a los objetivos del proyecto, así como los detalles de los algoritmos y de la ejecución.

Para implementer el modelo recomendador, primero debemos hacer un estudio del dataset. A continuación, haremos un análisis estadístico sobre las trazas analizadas, buscaremos correlacciones, y trataremos de extraer conocimiento para aplicar en el modelo.

5.2 Estudio de los datos

En esta sección nos familiarizamos con el conjunto de datos de las trazas de Nueva York y buscamos correlacciones que podamos aplicar en un modelo de recomendación.

El estudio se realiza sobre un portátil MacBook Pro, procesador 2.2 GHz Intel Core i7 y 16 GB 1600 MHz DDR3 de memoria. Contamos con las siguientes tecnologías:

- Apache Spark: Haremos el análisis en la consola interactiva basada en Python (PySpark), realizando transformaciones y peraciones sobre el RDD de Spark.
- SciPy[58]: Librerías en Python de ingeniería, matemáticas y estadística
- Plotly[59]: Librería en Python de visualización de datos (diagramas de dispersión, histogramas y mapas, entre otros).
    - Plotly está integrado con MapBox[60], un servicio online que permite realizar visualizaciones sobre mapas geográficos. Lo usaremos para la visualicación de las localizaciones.

Teniendo el ordenador en reposo, vemos que suele tener unos 10 GB libres. En consecuencia, configuramos Apache Spark para usar 10 GB.

5.2.1 Origen y formato

Los taxis de Nueva York se dividen principalmente en dos tipos:

- Taxis Amarillos o Medallón, que operan en los cinco distritos de la ciudad de Nueva York: Manhattan, Brooklyn, Queens, The Bronx, y Staten Island (Figura 37: Distritos de New York City).

- Taxis Verdes, que operan en Upper Manhattan, The Bronx, Brooklyn, Queens (excluyendo el aeropuerto LaGuardia y el aeropuerto internacional John F. Kennedy) y Staten Island.

Figura 37: Distritos de New York City

El NYC Taxi & Limousine Comission ofrece datasets para ambos tipos. Por requisitos del sistema, trabajaremos con los datos de los Taxis Amarillos. El dataset es actualizado cada cierto tiempo, y las trazas están divididas por meses. Cada mes tiene unos 10 millones de trazas de taxis.

Vemos que cada traza tiene múltiples atributos, pero para el proyecto nos centraremos en las siguientes propiedas:

- `tpep_pickup_datetime`: Fecha y hora inicial
- `tpep_dropoff_datetime`: Fecha y hora final
- `Passenger_count`: Número de pasajeros
- `Trip_distance`: Recorrido en millas
- `Pickup_longitude`: Coordenada longitud
- `Pickup_latitude`: Coordenada latitude

- `Total_amount`: Pago total

Los datos se recogen de forma automática por un dispositivo GPS instalado en el coche, excepto el número de pasajeros, que es introducido a mano por el conductor. Las propiedades completas las podemos ver en el Anexo B.

5.2.2 Correlacciones de características

Para explorar decidimos tomar una muestra aleatoria del 1% de las trazas de Enero de 2016. Si analizamos todas las trazas, el tiempo de computación haría imposible el análisis interactivo.

Lo primero que hacemos es tomar unas características o *features* en las que suponemos podemos extraer un valor predictivo que podamos aplicar en el modelo. Tras evaluar los campos disponibles, escogemos **tiempo**, **distancia**, y número de **pasarejos**.

Intuitivamente, podemos enunciar las siguientes hipótesis:

- Que el tiempo y la distancia estarán correlaccionadas (cuanta más distancia, más duración del viaje, y viceversa)

- Que cuanto mayor duración o distancia, esperamos mayor beneficio económico en un viaje en sí mismo, pero menor beneficio en su conjunto, dado que:

    - La probabilidad de que el taxista encadene más viajes se reduce

    - Hay más posibiliades de que el taxista acabe a las afueras, donde se da una menor frecuencia de viajes

- El número de pasajeros estará inversamente correlaccionada con tiempo y duración, dado que es más probable que se muevan grupos de personas juntas por el centro que en las afueras.

- Que cuanto mayor número de pasajeros, esperamos mayor beneficio económico en un viaje en sí mismo por el plus de pasajeros, y el mismo efecto en su conjunto.

Lo primero que hacemos es visualizar un diagrama de dispersion de las características. Si comparamos el tiempo, la distancia, y el número de pasajeros directamente con el pago total en cada traza, vemos que aparentemente sólo las dos primeras están correlaccionadas, y que el número de pasajeros es neutro. En la Figura 38. Correlacciones con el pago total vemos los diagramas de dispersión junto con la regresión linear, siendo β es el coeficiente de la regresión lineal, y r la correlación.

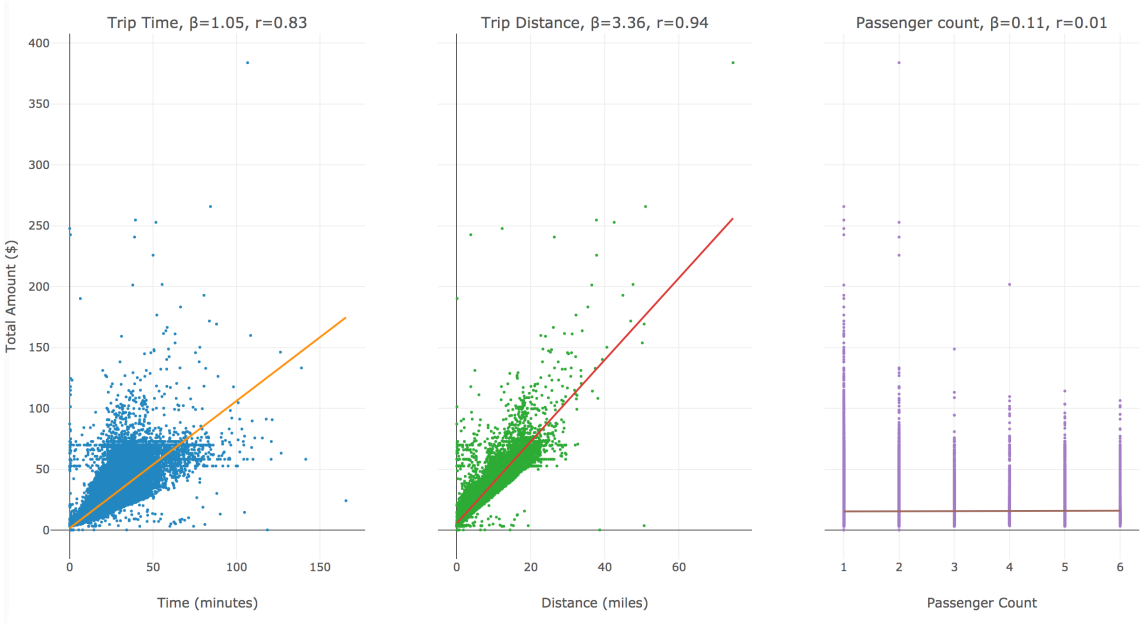


Figura 38. Correlacciones con el pago total

β es el coeficiente de la regresión lineal, y r la correlación.

Ahora bien, si comparamos el ratio de precio por cada característica, veremos que cuanto mayor sea la última, menor rendimiento económico por unidad ofrece. En la Figura 39. Correlación del rendimiento, vemos que hay una caída exponencial del rendimiento para las tres características. Es decir que cuanto menor sea la duración, distancia, y número de pasajeros, más rendimiento económico por minuto, milla y pasajero obtendremos, respectivamente.

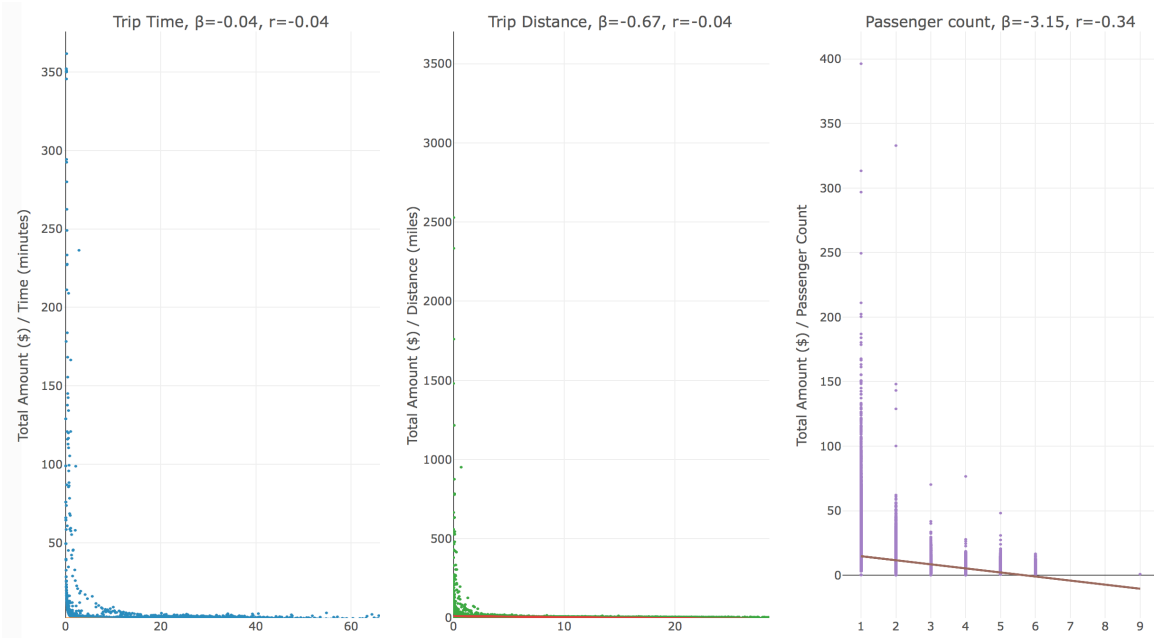


Figura 39. Correlación del rendimiento

β es el coeficiente de la regresión lineal, y r la correlación.

Hasta ahora hemos observando las trazas a nivel individual. Para terminar, si agrupamos las características a diferentes magnitudes y sumamos el coste total, podremos observar qué franjas obtienen mejores resultados.

Para ello, en la Figura 40. Correlación del dinero total producido, hemos agrupado las características por cada valor entero, truncando la coma flotante. Cualitativamente, podemos observar que, a menor magnitud, mayor beneficio total, en cualquiera de las tres características.

Vemos que las trazas que han llevado sólo un pasajero han sido las más productivas comparadas con las demás características. La segunda característica más significativa es la distancia, seguida por el tiempo.

Esto es importante, ya que en el modelo recomendador, lo que nos interesa es el beneficio en su conjunto, no en cada traza individual. Podemos sacar partido a las correlacciones encontradas en el modelo predictivo, puntuando las localizaciones basándonos en estas relaciones.

Tomamos nota del coeficiente de las regresiones lineales para usarlo posteriormente en el recomendador:

- Tiempo: $\beta = 422.52$
- Distancia: $\beta = 2728.12$
- Pasajeros: $\beta = 171616.23$

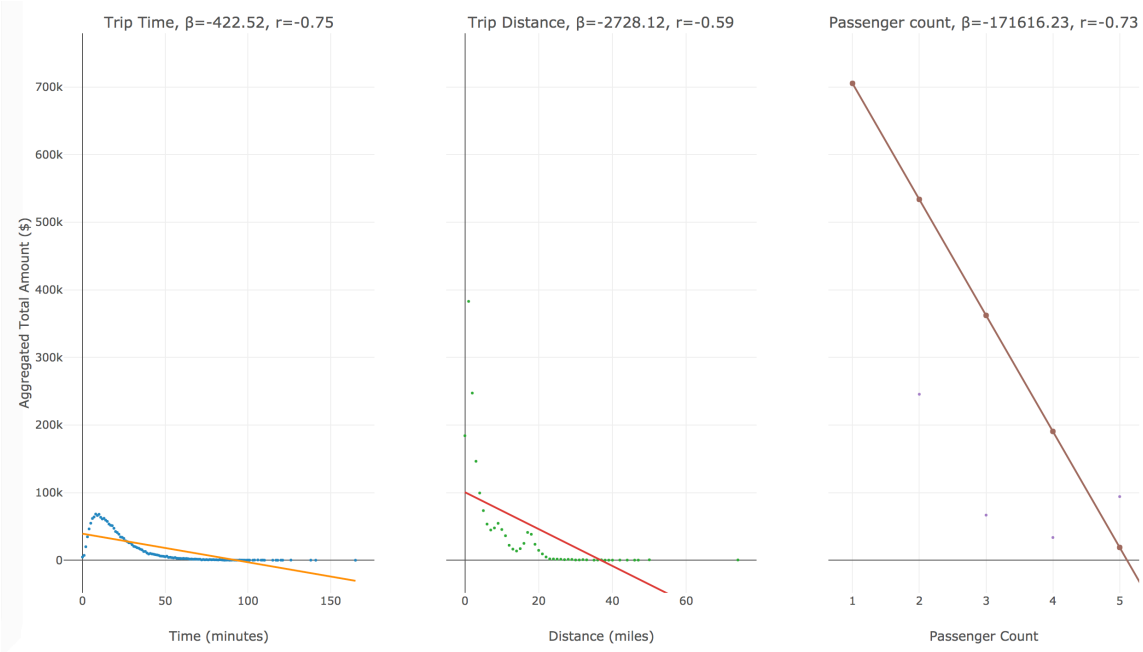


Figura 40. Correlación del dinero total producido

β es el coeficiente de la regresión lineal, y r la correlación.

Para concluir, nótese que todas las hipótesis que hemos formulado inicialmente son ciertas, excepto la última: cuanto menor número de pasajeros, mayor beneficio. Y ha resultado ser, sin duda la más importante, dado el coeficiente de la regresión lineal.

5.2.3 Frecuencias y clústers

Aquí analizaremos cómo cambian las frecuencias de los viajes con el tiempo y la localización.

Las localizaciones están compuestas por dos valores que representan la posición en un plano cartesiano:

- Longitud, el eje de ordenadas
- Latitud, el eje de abscisas

Basándonos en la fecha y la ubicación de recogida, vamos a tomar como características o *features* tres valores: la hora, el día de la semana, y la localización. Con ellas, construiremos las siguientes hipótesis:

- Que la frecuencia de los viajes en una ubicación dada cambia según la hora del día.
- Que la frecuencia de los viajes en una ubicación dada cambia según el día de la semana.
- Que la frecuencia de los viajes cambia de una ubicación a otra.

Para comenzar, en la Figura 41 visualizamos las localizaciones usando un diagrama de dispersión sobre el mapa del distrito Manhattan. Podemos detectar agrupaciones de trazas en diferentes lugares de recogida a simple vista. Una de las zonas más concurridas es el sur de Central Park.

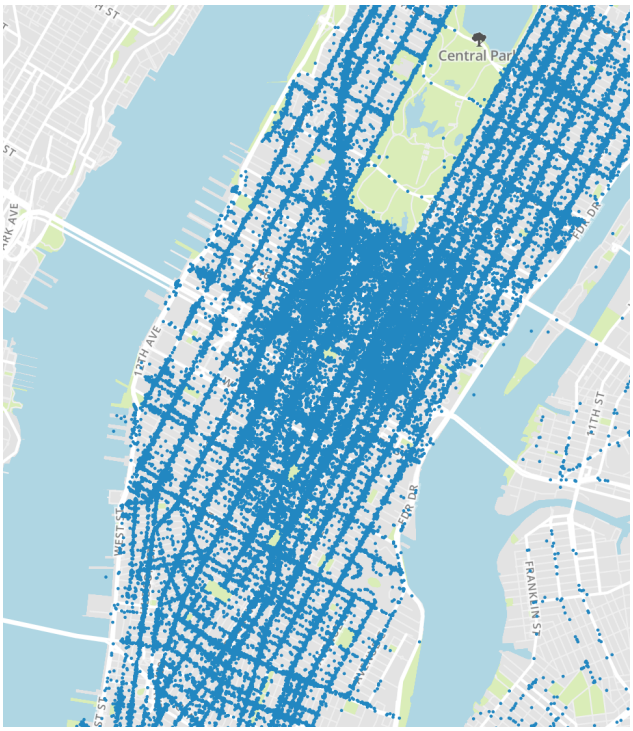


Figura 41. Diagrama de dispersión de coordenadas iniciales

Y si hacemos zoom, en la Figura 41. Diagrama de dispersión de coordenadas iniciales podemos ver que algunas agrupaciones son más más circulares, y otras más alargadas. También es posible observar que en algunas calles la densidad es tan alta que la agrupación toma la forma de la calle.

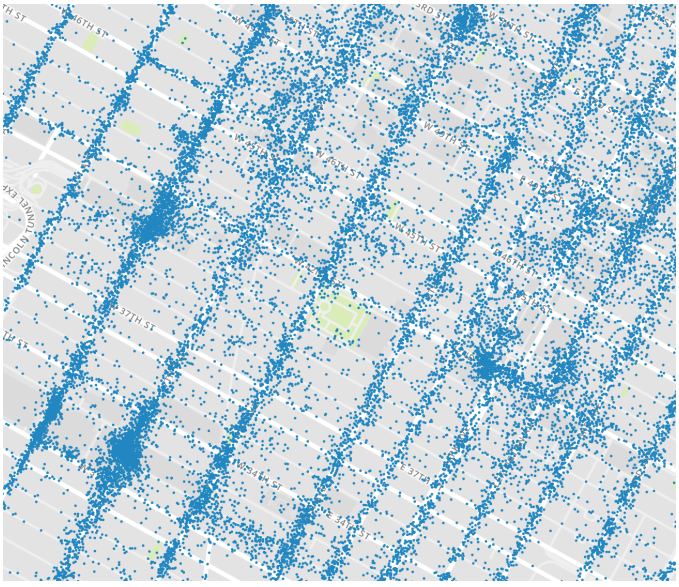


Figura 42. Detalle del diagrama de dispersión de coordenadas iniciales

Por otro lado, vamos a analizar las frecuencias de las trazas. Queremos ver si las frecuencias cambian a lo largo del día, y si también varían para diferentes días de la semana. Para ello, vamos a visualizar las frecuencias por horas de la semana, con una

serie para cada día de la semana. En la Figura 43 podemos apreciar que las frecuencias sí cambian. Además, los horarios de más actividad fluctúan entre las 8 de la mañana y las 12 de la noche, y la actividad cae especialmente a las 5 de la mañana en todos los días de la semana.

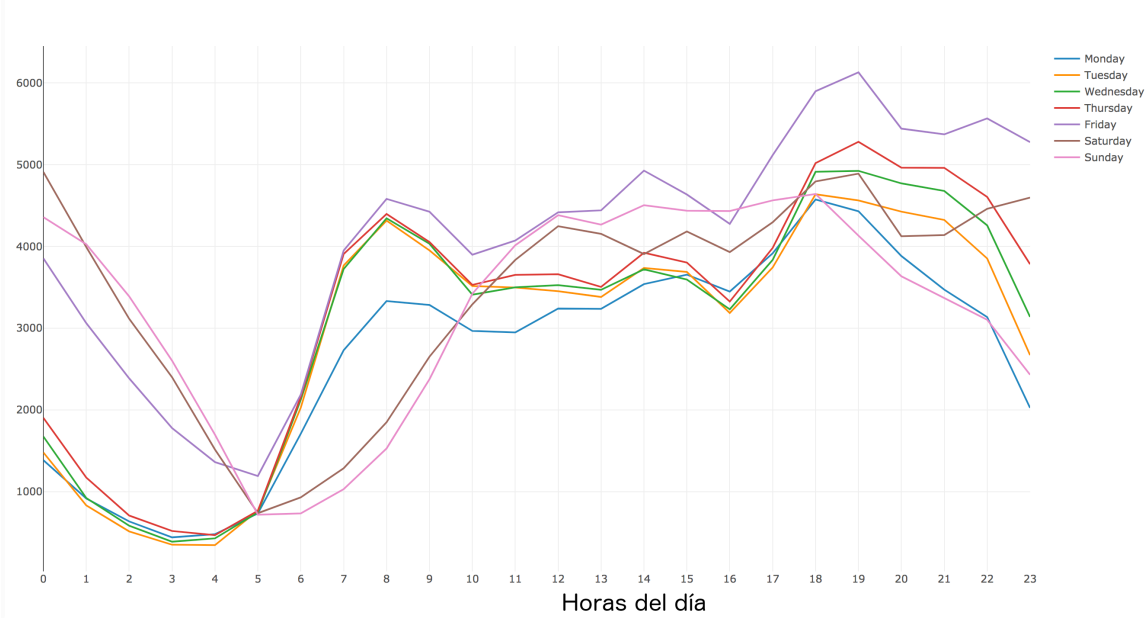


Figura 43. Frecuencia de viajes por hora del día

Es interesante señalar que:

- La noche de los lunes es similar a la de martes, miércoles y jueves.
- Martes, miércoles y jueves son prácticamente idénticos.
- Los viernes noche y sábado noche son muy parecidos.
- Los sábados y domingos son parecidos por la mañana, pero la noche del domingo es muy inferior.

Así, confirmamos la primera y segunda hipótesis: la frecuencia de los viajes cambia según la hora y el día de la semana.

En la Tabla 32: Correlación de frecuencias entre días de la semana listamos las correlacciones entre cada uno de los días, donde podemos valorar cuantitativamente las relaciones que acabamos de enumerar.

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Monday	1.0	0.97	0.96	0.95	0.89	0.52	0.49
Tuesday	0.97	1.0	0.99	0.98	0.89	0.42	0.33
Wednesday	0.96	0.99	1.0	1.0	0.92	0.46	0.33
Thursday	0.95	0.98	1.0	1.0	0.94	0.5	0.34
Friday	0.89	0.89	0.92	0.94	1.0	0.74	0.56
Saturday	0.52	0.42	0.46	0.5	0.74	1.0	0.89
Sunday	0.49	0.33	0.33	0.34	0.56	0.89	1.0

Tabla 32: Correlación de frecuencias entre días de la semana

Coefficiente *r* de correlación Pearson

El siguiente paso es confirmar que las agrupaciones cambian también con el día y la ahora. Vamos a tratar de encontrarlos mediante algoritmos de clustering.

Como hemos visto en el capítulo Estado del Arte, en clustering espacial para minería de datos DBSCAN es uno de los algoritmos más citados[61] en las publicaciones científicas. Pero no está disponible en la librería estándar de Spark. Las opciones disponibles son K-Means|| y el Gaussian Mixture Model (GMM). Las distribuciones observadas parecen gausianas, de modo que el GMM puede sernos útil.

Como estamos explorando de forma interactiva, dividimos el día en franjas de 4 horas y computamos los clusters para cada una de las franjas. De esta forma tendremos un número manejable de combinaciones para comparar de forma intuitiva, y además reducimos el tiempo de computación. En la Tabla 33 podemos apreciar que tenemos seis grupos distintos.

Grupo	Rango de horas
0	0-4
1	4-8
2	8-12
3	12-16
4	16-20
5	20-24

Tabla 33: Grupos de horas analizados

Recordemos que el modelo de clustering Gaussian Mixture requiere que especifiquemos el número de clústers (*k*) inicial, aunque luego pueda encontrar menos. Desconocemos el número de clusters que nos vamos a encontrar, pero sabemos que debe ser un número relativamente alto para cubrir todas las posibles agrupaciones de mayor densidad de las trazas. Escogemos arbitrariamente computar 1000 clústeres para cada franja horaria.

En la Figura 44 podemos ver los clústers obtenidos. Cada uno de los clústers es asignado un color arbitrario, y las localizaciones de las trazas son pintadas con el color del cluster al que pertenecen. Recordemos que el conjunto de los clústers cambia en cada franja horaria, por lo que aunque los colores de un mapa a otro coincidan, el clúster no es el mismo.

Se pueden apreciar que hay ligeras variaciones entre las zonas más frecuentes en cada momento del día. Al noreste del mapa, justo debajo del Central Park, podemos ver que la actividad aumenta especialmente por la tarde y la noche.

Y con esto confirmamos la tercera hipótesis: las frecuencias cambian dependiendo de la ubicación.

Figura 44. Clústeres obtenidos por franja horaria

5.3 Procesado de los datos

Los datos deben sufrir un proceso de trasformación y limpieza. Aquí trataremos de dar formato a los datos y desechar aquellos que no son válidos mediante reglas de negocio y manipulaciones requeridas por el sistema de destino. Los registros provenientes de los taxis no tienen transformaciones directas sobre los campos de la traza, pero se eliminan los registros considerados no válidos.

Durante el apartado anterior (5.2 Estudio de los datos) encontramos trazas tan imposibles como con localizaciones en mitad del mar, con 24 horas de tiempo recorrido, o con ningún pasajero, entre otros. Decidimos filtrar los datos de tal forma que cumplan las siguientes condiciones:

- Las coordenadas deben de estar dentro de las areas de los distritos en los que operan los taxis amarillos.
- La distancia recorrida debe ser mayor que cero y menor que 100 millas.
- El pago total debe ser mayor que cero.
- La fecha de recogida debe ser menor que la fecha final.
- El tiempo de recorrido debe ser inferior a 3 horas.

El filtrado se ha implementado sobre el RDD de Spark. Para más detalles sobre la implementación, por favor consultar el módulo “lib/process.py” en el Anexo C.

La comprobación de los puntos pertenecientes a los distritos no fue trivial. Se ha usado una distribución libre de los datos de los distritos en GeoJSON[62]. Posteriormente, la información es leída en Python con Descartes y Shapely[63]. Tanto el archivo como la lógica es compartida por los nodos del cluster. Por favor consultar el módulo “lib/boros.py” en el Anexo C para más detalles.

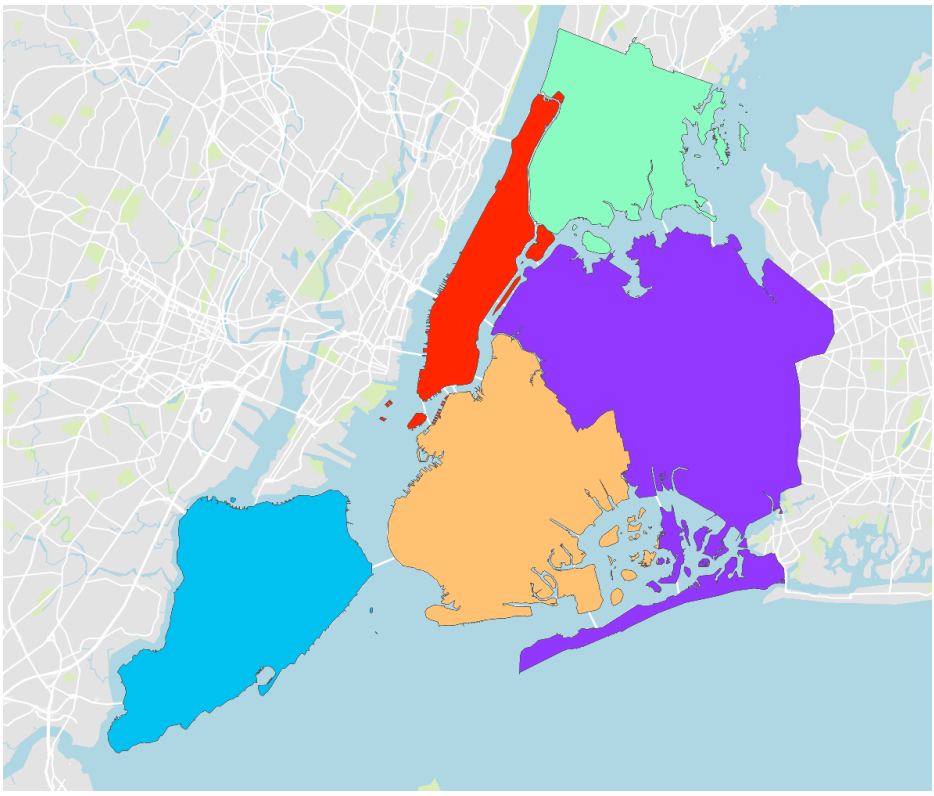


Figura 45. Distritos en GeoJSON

5.4 Modelo recomendador

En esta sección implementaremos el modelo que va a computar las recomendaciones.

5.4.1 Localización y área de las recomendaciones

En primer lugar, las recomendaciones tendrán una ubicación. Siguiendo el estudio realizado anteriormente, decidimos computarlo con el Gaussian Mixture Model (GMM) de Spark.

El siguiente reto que nos encontramos es conseguir una representación geométrica del clúster. Utilizando algoritmos de envolvente convexa (*convex hull)* podemos encontrar los vértices mínimos necesarios para representar los puntos de las agrupaciones de los clústers. SciPy incluye un módulo con el algoritmo conocido como *Quickhull*[64], el cual sigue los siguientes pasos:

1. Busca los puntos con coordenadas con latitud y longitud mínimas mínimas. Éstas siempre formarán parte del casco convexo.

2. Busca la línea formada por los dos puntos para dividir el conjunto en dos subconjuntos de puntos, los cuales serán procesados recursivamente.

3. Busca el punto, a un lado de la línea, con la distancia máxima desde la línea. Los dos puntos encontrados antes junto con éste forman un triángulo.

4. Los puntos situados dentro de ese triángulo no pueden ser parte del casco convexo y por lo tanto se ignorarán en los siguientes pasos.

5. Se repiten los pasos 3 y 4 sobre las dos líneas formadas por el triángulo (no sobre la línea inicial).

6. Se repite hasta que no se encuentran más puntos. Entonces la recursión ha llegado a su fin y los puntos seleccionados constituyen la envolvente convexa.

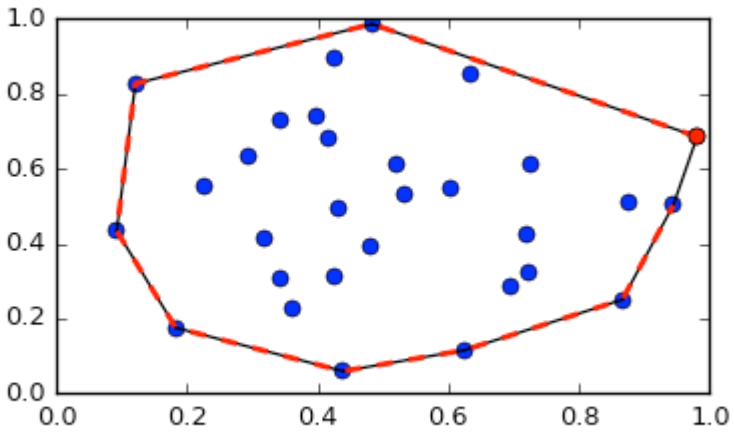


Figura 46: Ejemplo del resultado del algoritmo *Quickhull*

De esta forma, podemos representar los clústers como polígonos con vértices basados en latitud y longitud, estando ya geolocalizados.

5.4.2 Puntuación de las recomendaciones

El siguiente paso es encontrar una función que represente cuán lucrativo es un clúster, de tal forma que podamos ordenarlos y ofrecer las zonas más beneficiosas a los taxistas.

Decidimos dar una puntuación a cada uno de ellos basándonos en los coeficientes de las regresiones lineales computadas en el apartado 5.2.2, basándonos en la distancia, tiempo y número de pasajeros.

Siendo t el tiempo de la traza, d la distancia de la traza, y p los pasajeros de la traza, podemos computar S, la puntuación de la traza, siguiendo la función definida en la Figura 47.

$$S = 422.52\ t + 2728.12\ d + 171616.23\ p$$

Figura 47: Función de puntuación de recomendación

La puntuación total del cluster será igual a la suma de las puntuaciones de las trazas contenidas.

Se realiza un ensayo utilizando los clústers computados en el apartado 5.2.2 para comprobar el modelo. Así, en la Figura 48, podemos ver los polígonos computados para los diez mejores clusters de cada franja horaria utilizada en el estudio de los datos. Podemos distinguir algunas áreas más abiertas que otras. Muchos de las recomendaciones representan calles concretas.

Figura 48. Polígonos de los clusters con mayor puntuación

Cada leyenda indica la puntuación obtenida por el clúster

5.5 Aplicación Spark

En esta sección detallaremos el algoritmo seguido por la aplicación que será enviada a Spark para computar las recomendaciones.

Implementaremos el modelo en forma de un script en Python que Spark se encargará de ejecutar.

La estrategia a seguir con el planificador Spark es la siguiente:

1. Para cada día de la semana
    a. Para cada hora
        i. Filtrar el Dataset por día y hora
        ii. Computar los clústereres con GMM sobre los resultados filtrados
        iii. Obtener las trazas, cada una de ellas con el clúster asignado

A partir de aquí, la computación de los vértices de los polígonos, la computación de las puntuaciones totales de cada clúster, y la persistencia de los resultados, se realizan en el nodo localmente.

Para más detalles sobre la implementación, consultar el módulo "compute_clusters.py".

5.5.1 Traspaso de los resultados

Recordemos que los resultados serán volcados posteriormente en la base de datos PostGIS. Django ofrece una interfaz a la base de datos, así como una interfaz para línea de comandos. Aprovechando que la aplicación ejecutada en Spark está desarrollada en Python, los resultados se serializarán en un archivo mediante la librería *pickle*[65], disponible en la librería standard de Python y compartida por ambos procesos.

El formato escogido serán tuplas de tuplas siguiendo el formato de la Tabla 34.

Índice	Campo	Valor
0	Hora	Integer de 0 a 23
1	Día de la semana	Integer de 0 a 6 (siendo lunes el 0, y domingo el 6)
2	Polígono	Lista de pares de Float (longitud, latitud)
3	Puntuación	Puntuación del clúster

Tabla 34: Orden de los campos del resultado serializados

El script "import_recomendations.py" se encarga de deserializar el archio y volcarlo en la base de datos utilizando la interfaz de Django. Para más información, consultar el módulo "project/driver/taxi/management/commands/import_recommendations.py".

5.6 Base de datos espacial

Como hemos visto anteriormente, implementamos la base de datos en PostgreSQL junto con su extensión PostGIS.

Como hemos visto en el Estado del Arte, es una base de datos relaccional. El esquema constará de una única tabla, “taxi_recommendation”. En la Figura 49 podemos ver el detalle del SQL y en la Tabla 35 una descripción de los campos.

```
CREATE TABLE taxi_recommendation
(
    id integer NOT NULL DEFAULT nextval('taxi_recommendation_id_seq'::regclass),
    hour integer NOT NULL,
    weekday integer NOT NULL,
    score double precision NOT NULL,
    poly "public.geometry"(1107464) NOT NULL,
    CONSTRAINT taxi_recommendation_pkey PRIMARY KEY (id)
)
WITH (
    OIDS = FALSE
)
TABLASPACE pg_default;

CREATE INDEX taxi_recommendation_poly_id
    ON public.taxi_recommendation USING gist
    (poly)
    TABLASPACE pg_default;
```

Figura 49: SQL de la tabla "taxi_recommendation"

Campo	Descripción
id	Identificador de la recomendación
hour	Hora del día
weekday	Día de la semana
score	Puntuación del clúster
poly	Polígono que lo representa

Tabla 35: Campos de la tabla “taxi_recommendation”

Nótese que se indexan los polígonos para que las consultas por localización seán mas rápidas.

Sin embargo, la implementación real se ha realizado con la abstracción ofrecida por la interfaz de Django. De esta forma, el esquema está distribuido en dos ficheros distintos:

`project/driver/taxi/models.py`
Define el nombre y los atributos del modelo.

`project/driver/taxi/migrations/0002_recommendation.py`
Define la operación de creación de la tabla. Es inferida automáticamente por Django a partir del modelo definido anteriormente.

Django permite abstraerse de las relaciones a través de su interfaz conocida como ORM (*Object Relational Mapper*), la cual permite interactuar con las filas de las tablas como si fuesen objetos de Python.

5.7 Aplicación Web

Para la aplicación web, como vimos en el estado del arte, escojemos las siguientes tecnologías debido a que el autor ya tiene experiencia con ellas:

- PostgreSQL + PostGIS para la base de datos espacial
- Django (librería en Python) como web framework

No esperamos un tráfico muy alto, por lo que consideramos razonable desplegar todos los componentes en un solo VPS de mínimas características. Escogemos la opción más básica en DigitalOcean[66]:

- 512 MB de RAM
- 1 vCPU
- 20GB de disco duro SSD
- 1TB de transferencia

La implementación del formulario se ha realizado también en Django, utilizando su sistema de formularios. Las entradas del formulario son:

- Día de la semana (por defecto, día de la petición)
- Hora (por defecto, hora de la petición)
- Longitud
- Latitud

Al recibir la petición por el formulario, el sistema hace una consulta buscando las recomendaciones a 10 millas a la redonda de la ubicación dada, ordenando los resultados por puntuación descendiente, y limitando el número de resultados a 10. En la figura

podemos ver la consulta que realiza la aplicación a la base de datos, tomando los parámetros como entrada, utilizando la interfaz de base de datos de Django.

```
        hour = params.get('hour')
        weekday = params.get('weekday')
        lat = float(params.get('lat'))
        lng = float(params.get('lng'))

        point = Point(lng, lat, srid=4326)
        recommendations = (
            Recommendation.objects
            .filter(hour=hour, weekday=weekday)
            .annotate(distance=Distance('poly', point))
            .annotate(point=Func(
                F('poly'),
                Func(Value(str(point)), function='ST_PointFromText'),
                function='ST_ClosestPoint')
            )
            .filter(distance__lte=measure.Distance(mi=5).m)
        ).order_by('-score')[:10]
```

Figura 50: Consulta de recomendaciones a la base de datos

Tras resolver la consulta, la aplicación web pinta los resultados situando las recomendaciones geográficamente sobre un mapa creado con la API con Google Maps[67]. El mapa muestra la ubicación dada y las recomendaciones disponibles a 10 millas alrededor, indicando cada una su importancia con el icono sobre el mapa. Al hacer click sobre cualquiera de ellas, se redirigire a Google Maps con las coordenadas de la recomendación para que el conductor pueda poner el navegador.

Para más detalles de la interfaz gráfica, véase Apéndice C.

Capítulo 6: Pruebas y resultados

6.1 Pruebas funcionales en PC Portátil

Primero se realizará una prueba funcional en un entorno doméstico. Se ejecutará el sistema completo sobre el 1% de las trazas de Enero de 2016 para comprobar que funciona. Se ejecutará sobre un MacBook Pro, con procesador 2.2 GHz Intel Core i7, 16 GB 1600 MHz DDR3 de memoria, y disco duro SSD.

```
$ spark/bin/spark-submit compute_clusters.py
```

Figura 51: Comando de ejecución del sistema en el PC portátil

La ejecución se resuelve favorablemente. Las recomendaciones se computan en 4 horas, y su carga en la base de datos tarda menos de 4 segundos.

```
$ ./manage.py import_recommendations
```

Figura 52: Comando de carga de resultados

Tras cargarse las recomendaciones, hacemos una consulta en la aplicación web. Podemos ver un ejemplo de captura de la aplicación web en la Figura 53. Aplicación web tras la carga de las recomendaciones computadas.

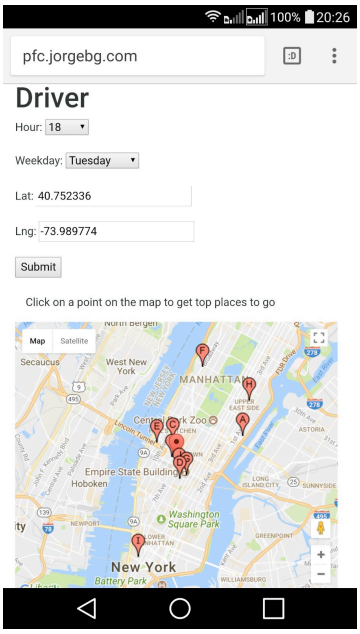


Figura 53. Aplicación web tras la carga de las recomendaciones computadas

6.2 Pruebas de rendimiento en clúster universitario

En esta sección ejecutaremos las pruebas de rendimiento y evaluaremos los resultados.

Se dispone del cluster universitario, compuesto por seis servidores (conocidos como *monitor01*, *monitor02*, …, *monitor06*), prestados por el departamento de Ingeniería Telemática. Tienen las siguientes características:

- Procesador Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz
- 16 GB de RAM
- Disco duro con 30GB de cuota, compartido por NFS

Considerando que son máquinas compartidas, se va a evitar sobrepasar los 10GB de uso de memoria RAM.

6.2.1 Rendimiento de GMM

Primero se realizan las pruebas de rendimiento del algoritmo de clusterización de modelo Gaussian Mixture. Dichas pruebas se ejecutan sobre un sólo nodo.

```
$ spark/bin/spark-submit --py-files lib.zip test_gaussian.py
```

Figura 54: Comando de ejecución de pruebas de rendimiento de GMM

Se prueba el modelo GMM con diferentes entradas:

- Números de registro: 1, 10, 100, 1000, 10000 y 100000.
- Número de clústers a computar (*k*): 5, 25, 50, 100, 500 y 1000.

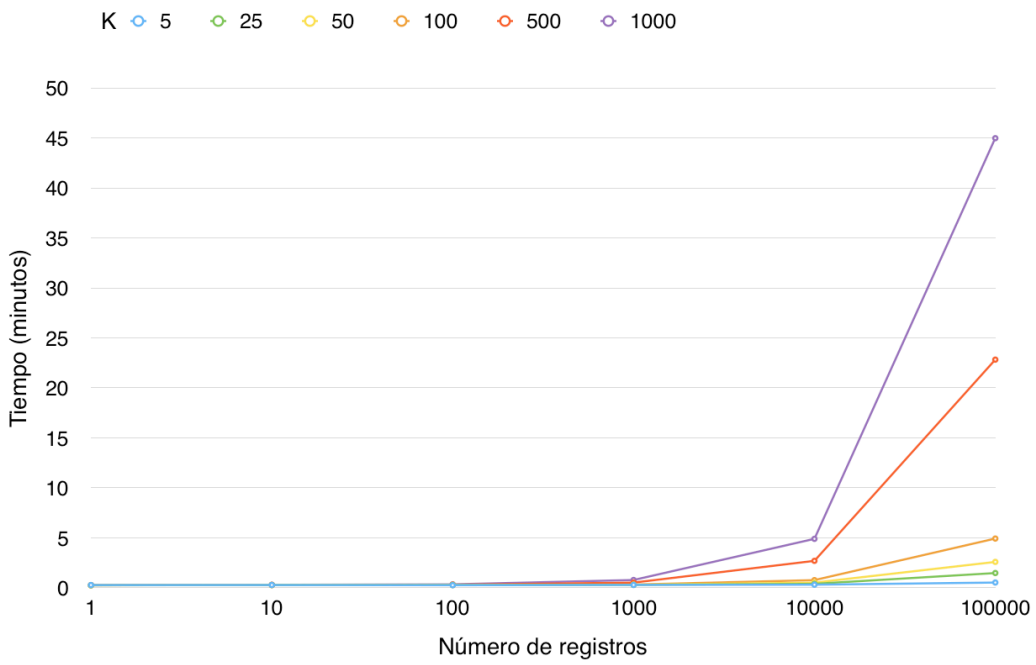


Figura 55: Tiempos de cálculo de clústers con GMM

k	*5*	*25*	*50*	*100*	*500*	*1000*
n						
1	0.27	0.25	0.25	0.26	0.26	0.26
10	0.28	0.28	0.28	0.28	0.29	0.30
100	0.26	0.28	0.28	0.29	0.31	0.35
1000	0.29	0.30	0.30	0.33	0.52	0.78
10000	0.31	0.40	0.52	0.77	2.70	4.90
100000	0.53	1.48	2.59	4.93	22.82	44.98

Tabla 36: Tiempos de cálculo de clústers con GMM

En la Figura 55 podemos ver los diferentes tiempos que le ha llevado al programa computar los clústers para cada combinación de entradas. Vemos que los tiempos no comienzan incrementar significativamente hasta que no se procesan mil registros. Con un número de registros inferior a mil, el número de clústeres k no afecta prácticamente al tiempo de cómputo.

Sin embargo, a partir de mil registros de entrada podemos ver que el incremento de k multiplica los tiempos totales de cómputo.

- Con 10^4 registros, sólo son notables para las k=500 y k=1000
- Con 10^6 registros, todas las k mayores que 5 alargan el cómputo, especialmente para k=500 y k=1000.

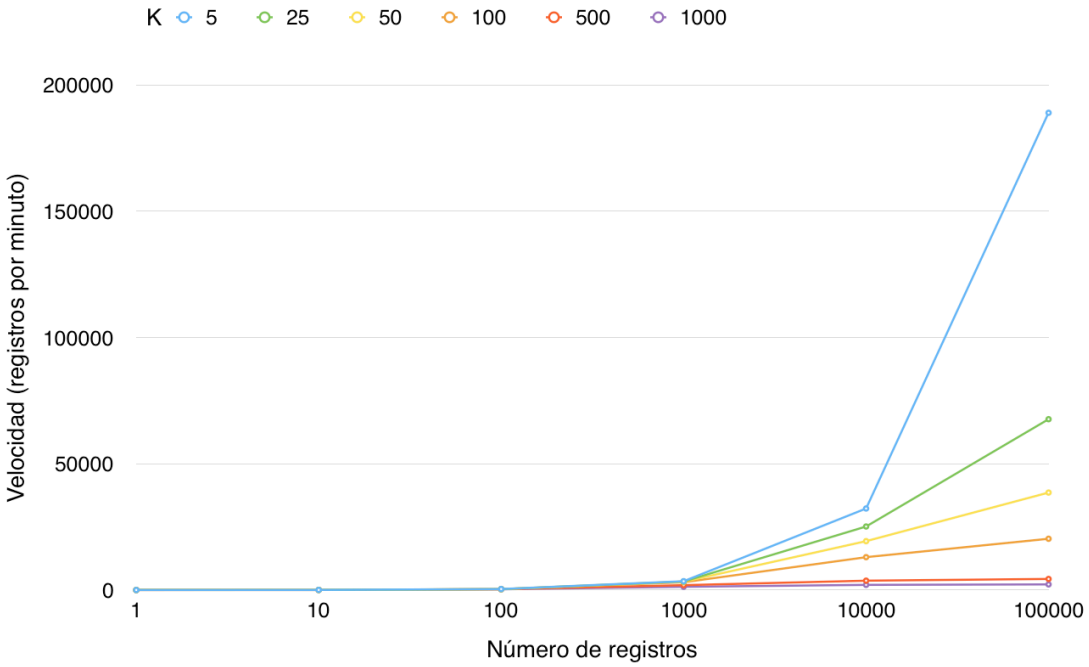


Figura 56: Velocidad de cálculo de clústers con GMM

k	*5*	*25*	*50*	*100*	*500*	*1000*
n						
1	3.65	3.92	3.96	3.92	3.92	3.78
10	35.63	35.66	36.12	35.86	34.78	33.27
100	386.40	354.28	355.65	349.80	323.40	289.48
1000	3481.17	3371.40	3305.71	3076.12	1926.85	1282.03
10000	32281.08	25148.22	19346.43	13010.66	3708.61	2042.01
100000	188957.53	67640.77	38589.05	20297.63	4381.28	2223.33

Tabla 37: Velocidad de cálculo de clústers con GMM

Viendo la influencia de la magnitud de *k* en los tiempos, es de esperar que tenga un efecto inverso en la velocidad. En la Figura 56 podemos ver las diferentes velocidades que le ha llevado al programa computar los clústers para cada combinación de entradas. Vemos que, al igual que cuando analizamos el tiempo total, las velocidades no comienzan a incrementar significativamente hasta que no se procesan mil registros. Con un número de registros inferior a mil, el número de clústeres *k* no afecta prácticamente al tiempo de cómputo.

Sin embargo, a partir de mil registros de entrada podemos ver que el incremento de *k* divide los tiempos totales de cómputo.

- Con 10^4 registros, sólo son notables para las *k*=5, *k*=25, *k* =50, y *k* =100. Para *k=500 y k=1000* la diferencia es despreciable.
- Con 10^6 registros, todas las *k* menores que 500 aumentan la velocidad de cómputo, especialmente para *k*=5.

6.2.2 Escalabilidad del sistema

Se procede a ejecutar la computación total de recomendaciones para 24 horas de un sólo día. El objetivo es evaluar la escalabilidad del sistema. Se van a ejecutar las pruebas en múltiples iteraciones probando diferentes configuraciones de máquinas para el clúster Spark:

- 1 Maestro, 1 Esclavo
- 1 Maestro, 2 Esclavos
- 1 Maestro, 3 Esclavos
- 1 Maestro, 4 Esclavos
- 1 Maestro, 5 Esclavos

El maestro es siempre ejecutado sobre el servidor *monitor01*.

Para cada una de estas iteraciones, se van a probar las siguientes configuraciones:

- Por defecto: 1 Worker por esclavo, 10GB
- Múltiple: 2 Workers por esclavo, 5 GB cada uno, es decir, 10GB en total

Las configuraciones se administran por medio del módulo **`fabfile.py`**. Dicho módulo define una serie de tareas que serán ejecutadas por el programa Fabric. A continuación, se muestra una secuencia de comandos de ejemplo para configurar el servidor con dos esclavos por defecto (un trabajador por esclavo).

```
$ fab config:settings=default,slaves=2
```

Figura 57: Comando de configuración del clúster

```
$ fab cluster:start
```

Figura 58: Comando de arranque del clúster

```
$ ssh monitor01 "spark/bin/spark-submit compute_clusters.py"
```

Figura 59: Comando de ejecución de aplicación en el clúster

```
$ fab cluster:stop
```

Figura 60: Comando de parada del clúster

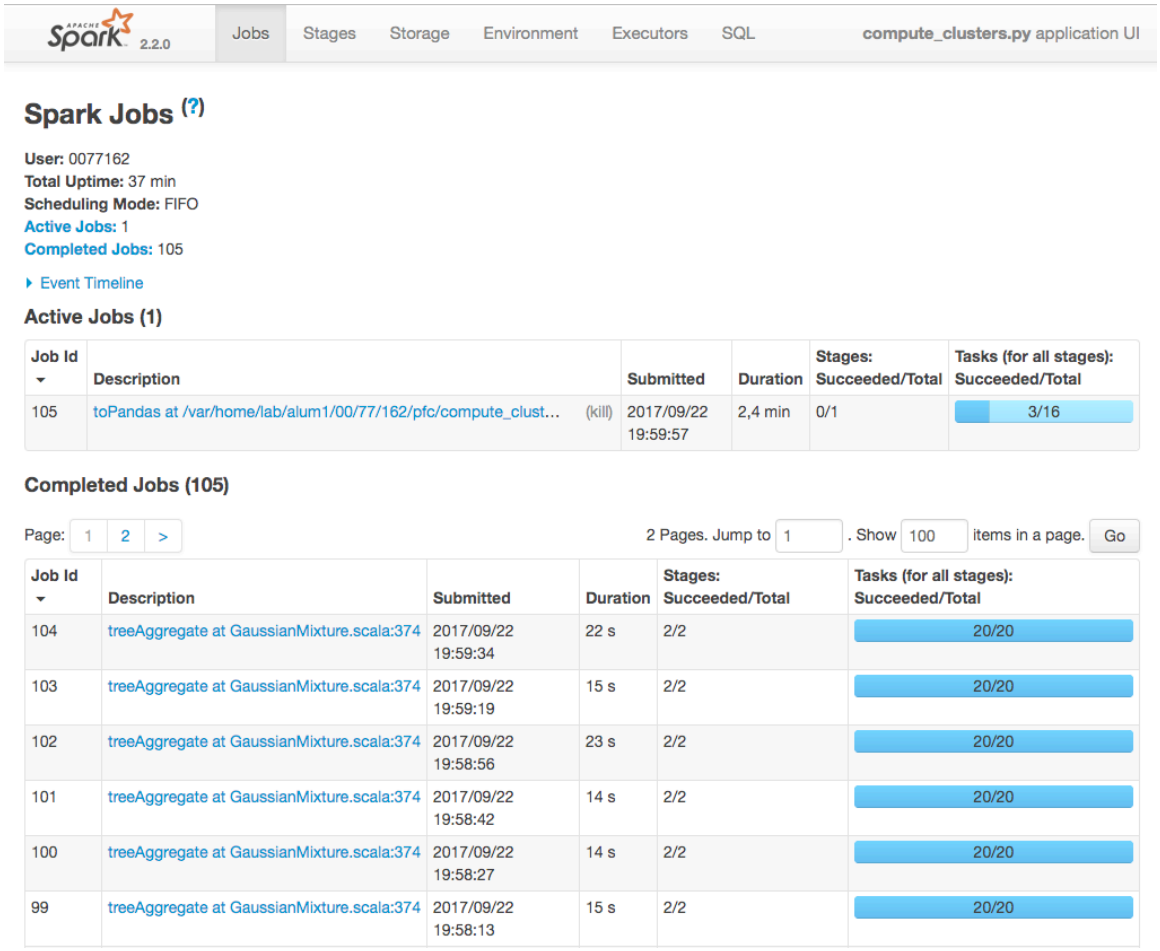


Figura 61: Captura de la UI de administración del cluster Spark

Se repite la secuencia de comandos para cada una de las configuraciones. Se evaluarán dos métricas diferentes:

- Tiempo de lectura de ficheros
- Tiempo de computación de las recomendaciones

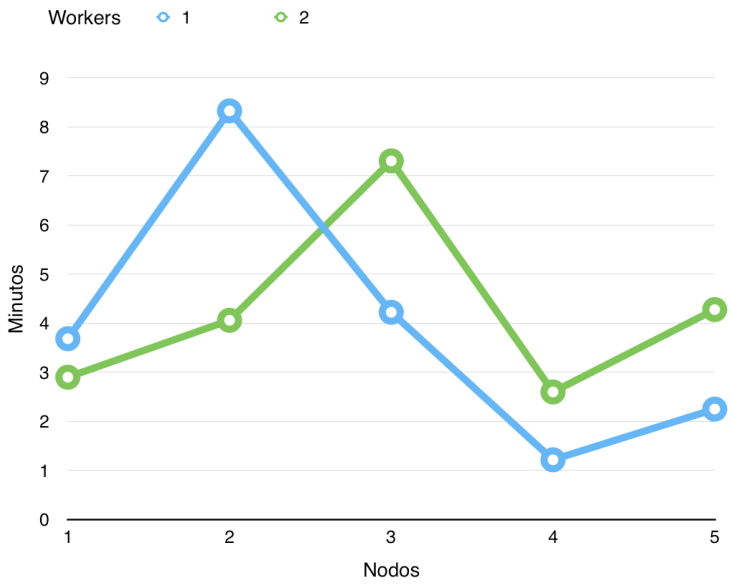


Figura 62. Tiempo de lectura del fichero en el clúster

En la Figura 62 podemos ver el tiempo de lectura del fichero. En general, el tiempo óptimo de lectura se ha conseguido con las configuraciones de cuatro y cinco servidores, con un solo trabajador en cada uno de ellos.

El aumento en el tiempo de cómputo que se da tanto en la configuración de un solo trabajador como en la configuración de dos trabajadores es de esperar, dado que el sistema tiene que buscar un equilibrio entre la latencia introducida por la comunicación entre los nodos y la reducción del tamaño de las particiones a distribuir entre los trabajadores.

Para la configuración de un solo trabajador, dos nodos tardan en leer los ficheros el doble que uno solo. No es hasta la incorporación del tercer nodo que los tiempos se igualan, y se alcanza la configuración optima con el cuarto nodo. El tiempo obtenido al añadir el quinto nodo es similar al obtenido con el cuarto, y ambos son menores que el tiempo de lectura de un solo nodo.

Para la configuración de dos trabajadores, el tiempo no deja de ascender hasta que se incorpora el cuarto nodo, a diferencia de la configuración de un solo trabajador, en cuyo caso el descenso empieza en el tercero. Sin embargo, continuando con la comparación con la configuración singular, el tiempo es menor cuando se usan dos trabajadores en un mismo nodo, que si se usa uno solo. Por último, el numero óptimo de nodos para esta configuración es cuatro.

Concluimos pues que la mejor configuración ha sido cuatro nodos con un solo trabajador por nodo.

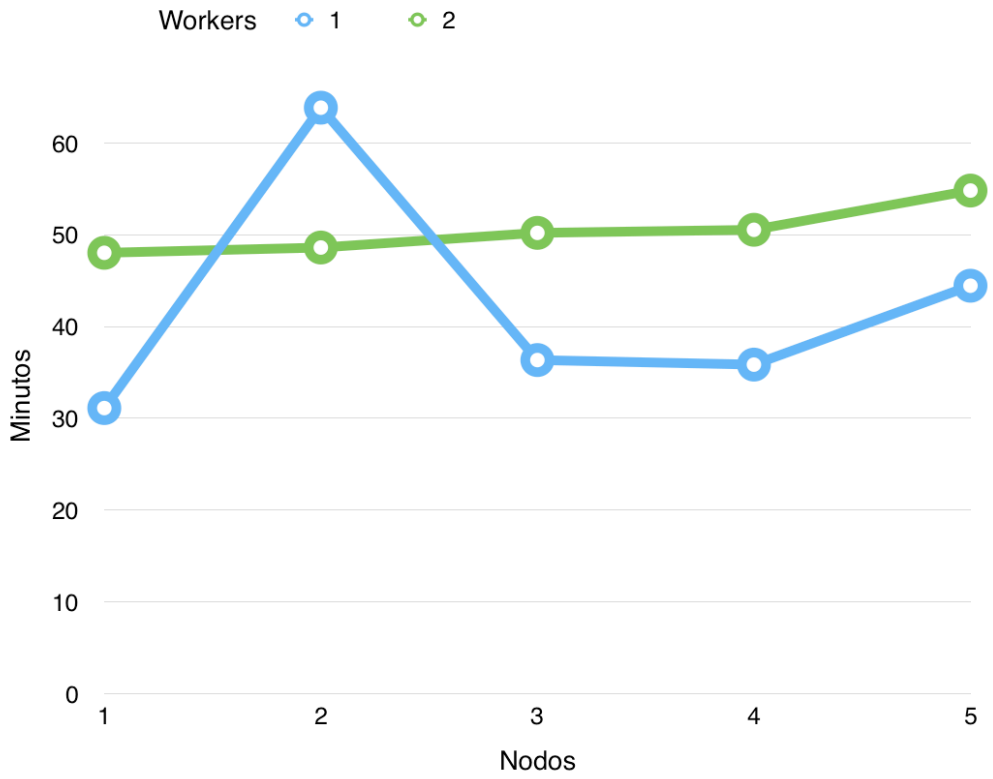


Figura 63. Tiempo de computación de recomendaciones en el clúster

En la Figura 61 podemos ver el tiempo de computación de las recomendaciones en el clúster. El clúster no escala de forma eficiente. Ninguna de las configuraciones ha sido capaz de mejorar los tiempos de un solo nodo con un solo trabajador.

Excepto para la configuración con dos nodos, el resto de los casos la configuración de dos trabajadores por nodo ha rendido peor que la de un solo trabajador.

Al igual que en las métricas sobre los tiempos de lectura del fichero, se esperaba un incremento del tiempo inicial, para luego encontrar un punto de ruptura en el que se encontrase un equilibrio entre la capacidad de cómputo añadida y la latencia introducida por la comunicación entre los nodos. Pero dicho equilibrio no se ha dado.

El sistema podría tener un punto de equilibrio más tardío de lo que esperábamos. El algoritmo de clusterizacion GMM actualiza los valores de los centroides de forma distribuida en cada iteración, y en consecuencia los nodos deben compartir la información computada en cada iteración. Esto podría producir un estrés tal en la comunicación entre los nodos que haría inviable la escalabilidad con el número de nodos usado[68].

Concluimos que la computación de las recomendaciones no escala.

Capítulo 7: Conclusiones y futuras líneas de trabajo

7.1 Conclusiones

El objetivo de este Trabajo de Fin de Carrera era el desarrollo de un sistema *big data* que ayudase a los conductores de taxis de Nueva York a ser más competitivos, tratando de fortalecer sus ventajas orgánicas frente a los servicios de transporte privado de economía colaborativa con ayuda de la tecnología.

Hemos desarrollado el sistema recomendador completo. Por un lado, el procesamiento de datos y la extracción de conocimiento. Por otro, la aplicación web para que los taxistas puedan consultar las recomendaciones.

Por tanto, el sistema de recomendación cumple con el objetivo principal, pero tras las pruebas de rendimiento sabemos que no es escalable. No hemos logrado una mejora significatia en el rendimiento al incrementar el número de nodos en el clúster.

7.2 Futuras líneas de trabajo

A continuación, se identifican y proponen diferentes líneas de trabajo un futuro proyecto a partir de éste.

7.2.1 Escalabilidad del cluster

Como hemos visto en el capítulo de Pruebas y Resultados, la computación de las recomendaciones no es escalable. Como futura línea de investigación, se podrían buscar algoritmos alternativos que tenga una exigencia de comunicación entre los nodos menor que GMM, de tal forma que se reduzca la latencia.

También se podría intentar encontrar el punto de ruptura en el que añadir nodos al cluster empieza a mejorar el rendimiento, probando configuraciones con más de seis nodos, que es el número de nodos probados en este proyecto.

7.2.2 Validación del modelo

El siguiente paso sería hacer un estudio para validar que el modelo incrementa los beneficios de los taxistas que lo usan.

La forma más sencilla desde el punto de vista estadístico es hacer un estudio de campo, pero requiere de muchos recursos para encontrar los candidatos, realizar las

pruebas y recoger los datos. Con una muestra lo suficientemente grande, dividida en dos grupos, un grupo de control y otro de intervención, podríamos ver si produce efecto. Y si lo produce, podemos cambiar los parámetros y las funciones de las puntuaciones y repetir el experimento dividiendo más grupos.

La validación del modelo En el capítulo Estado del Arte, vimos dos publicaciones científicas que habían usado dicho método. Uno de ellos usó un método híbrido, añadiendo al estudio de campo simulaciones de trazas con fuentes de datos diferentes.

7.2.3 Mejoras del modelo

Recordemos que la función que calcula la puntuación está basada en las regresiones lineales de las correlacciones encontradas durante el estudio de los datos. Se tomaron el coeficiente de la regresión lineal como indicador de puntuación, pero las curvas están lejos de ser lineales. Podemos mejorar el modelo buscando las funciones de las curvas y combinándolas en una nueva función de puntuación de traza.

Por ejemplo, en la Figura 64. Polinomio de grado 9 y curva agregada de tiempo vemos un polinomio de grado 9 encajaría muy bien para predecir la característica "tiempo".

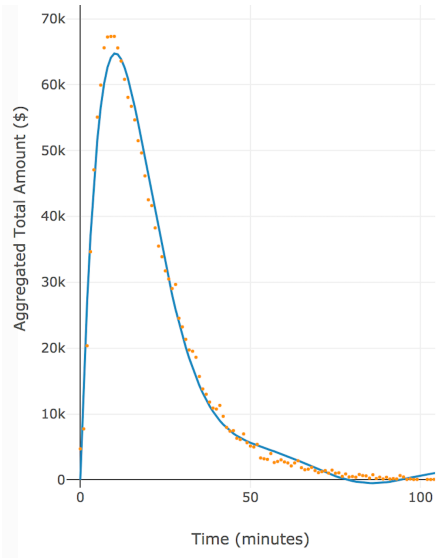


Figura 64. Polinomio de grado 9 y curva agregada de tiempo

Por otro lado, hemos probado con un número de clusters inicial *k* para el algoritmo GMM fijado en 1000. Otras formas de mejorar el modelo serían probar diferentes números. No sólo para la calcular los mejores clusters, sino también para optimizar el rendimiento del sistema. Es posible que un número menor ofrezca resultados de igual valor, a un coste de computación menor.

7.2.4 Evaluación de los riesgos del modelo

El uso masivo del recomendador en si mismo podría impactar en los taxis de varias formas:

- Saturación, al ir tantos taxis para el mismo sitio pueden provocar tráfico repentino en la zona.

- Las zonas recomendadas podrían dejar de ser rentables al quedarse sin clientes a los que coger debido al exceso de taxistas.

- Otras zonas podrían sufrir una carencia de taxistas debido a que no aparezcan recomendadas en el sistema.

Capítulo 8: Planificación

8.1 Fases de desarrollo

La duración de la realización del proyecto ha sido alrededor de tres meses. A continuación se resumen las fases en las que se ha dividido el desarrollo y el tiempo empleado para cada una de ellas:

1. Planteamiento de la necesidad y descripción del problema

En esta fase se pretende centrar la idea general del proyecto. Se plantea la necesidad de probar nuevas tecnologías y se acuerda cuáles se van a utilizar en concreto para el desarrollo.

- Tiempo estimado: 5 días.
- Participantes: Tutor y desarrollador del proyecto.

2. Estudio de las tecnologías utilizadas

Análisis de las tecnologías que se van a emplear. Se hizo un estudio de Spark.

- Tiempo estimado: 20 días.
- Participantes: Tutor y desarrollador del proyecto.

3. Análisis de los datos
Se exploran los datos y se evalúa la viabilidad.

- Tiempo: 30 días.
- Participantes: Tutor y desarrollador del proyecto.

4. Diseño del sistema

Fase en la que se definen los requisitos del sistema y la arquitectura de componentes que lo forman, así como la interacción entre ellos y la elección del diseño final.

- Tiempo: 5 días.
- Participantes: Tutor y desarrollador del proyecto.

5. Implementación del sistema

Es la fase en la que se implementa el diseño.

- Tiempo: 10 días.
- Participantes: Tutor y desarrollador del proyecto.

5. Pruebas y resultados

Pruebas de validación del correcto funcionamiento del sistema en diferentes entornos. También se miden tiempos de ejecución y se contrastan los resultados.

- Tiempo: 10 días.
- Participantes: Tutor y desarrollador del proyecto.

6. Redacción de la memoria

Documentación final de todo el trabajo realizado en el desarrollo completo del proyecto.

- Tiempo: 5 días.
- Participantes: Tutor y desarrollador del proyecto.

Capítulo 9: Presupuesto

9.1 Medios Empleados

Los recursos empleados en el desarrollo del proyecto son los siguientes:

Recursos humanos

- 1 Ingeniero Junior
- 1 Ingeniero Senior

Recursos materiales

- 1 PC portátil de gama alta
- Microsoft Office 365

Otros recursos

- Conexión a Internet durante 3 meses
- 6 Servidores de gama alta (cluster universitario)
- VPS Digital Ocean (aplicación web)
- Software
    - Django
    - Nginx
    - PostgreSQL, PostGIS
    - SciPy
    - Sklearn
    - Apache Spark

9.2 Presupuesto del trabajo

Autor: Jorge Barata González

Departamento: Ingeniería Informática

Descripción del Proyecto:

- Título: Sistema Recomendador de Taxis para Big Data.
- La duración del proyecto por el que se ha realizado el presupuesto es de 3 meses. Se ha contabilizado por unos costes indirectos del 20%.

El presupuesto total del proyecto durante los 3 meses es de **12.336€**.

El presupuesto lo he dividido en dos apartados: costes directos y costes indirectos.

- Los costes directos son los que se asocian con el producto, en mi caso incluyo el gasto del personal necesario para la realización del proyecto y el material utilizado para su desarrollo.
- Los costes indirectos los he presupuestado como un 20% del coste total del proyecto. Estos costes indirectos incluyen los gastos que no son directos del producto.

9.2.1 Costes directos

Los costes directos se desglosan a continuación.

Personal

El personal necesario para el correcto funcionamiento del Sistema Recomendador de Taxis para Big Data es de dos personas con titulación de Ingeniero Informática con un nivel Senior.

El presupuesto de este personal se ha calculado mediante el Convenio colectivo nacional de empresas de ingeniería y oficinas de estudios técnicos, el correspondiente a trabajadores con categoría de ingenieros informáticos.

He calculado el coste según los días en que estos trabajadores han realizado el proyecto. Uno de ellos ha trabajado durante media jornada, el otro un 10%.

TAREAS	DEDICACIÓN TOTAL (DÍAS)	DEDICACIÓN INGENIERO JUNIOR (HORAS)	DEDICACIÓN INGENIERO SENIOR (HORAS)
Planteamiento de la necesidad y descripción del problema.	5	20	4

Estudio de las tecnologías utilizadas.	20	80	16
Análisis de los datos.	10	40	8
Diseño del sistema.	5	20	4
Implementación del Sistema.	10	40	8
Pruebas y resultados.	10	40	8
Redacción de la memoria.	25	100	20
Total	**85 días**	**340 horas**	**68 horas**

Tabla 38: Horas de desarrollo

CONCEPTO	HORAS DEDICADAS	COSTE POR HORA	COSTE TOTAL
Junior	340	18€	6.120€
Senior	68	30€	2.040€
		TOTAL	**8.160€**

Tabla 39: Coste de desarrollo

Material

Para este proyecto el material necesario ha sido un ordenador portátil. El resto ha sido facilitado por el Departamento de Ingeniería de la UC3M para poder desarrollar el proyecto.

Concepto	Coste (€)	Dedicación	Período de depreciación	Coste imputable (€)
Ordenador portátil	2.000€	3 meses	48 meses	125€

Tabla 40: Coste de material

Otros costes

En otros costes incluimos la conexión a internet y el servicio VPS.

Concepto	Cantidad	Dedicación	Coste por hora (€)	Coste total (€)
Conexión a internet	1	3 meses	0.027€	60€
DigitalOcean VPS	1	1 año	0.027€	60€

Tabla 41: Otros costes

Total

El presupuesto total sumando:

- El personal necesario para realizar el proyecto.
- El material que se ha utilizado.

Personal	8.160€
Material	2.000€
Otros	120€
Total	**10.280€**

Tabla 42: Coste total

9.2.2 Costes Indirectos

A los cálculos anteriores habría que aumentarle los costes indirectos correspondientes al proyecto. Se han calculado en un 20% del presupuesto total. Es decir, este coste sería de **2.032€.**

Coste Total

El coste total del proyecto sería la suma de los costes directos y los costes indirectos.

Costes directos	10.280€
Costes indirectos	2.056€
Total	**12.336€**

Tabla 43: Coste total con impuestos indirectos

Capítulo 10: Anexos

Anexo A: Capturas Aplicación Web

Podemos visitar la aplicación en http://pfc.jorgebg.com.

De cara al usuario final, la web sólo tiene una página (Figura 65. Página principal). Se inicia automáticamente con la hora y día actuales, y un punto arbitrario de la ciudad de Nueva York, junto con los lugares recomendados. La prioridad de las recomendaciones las dan las letras en los iconos, ordenadas alfabéticamente.

Pueden cambiarse los parámetros y consultar las recomendaciones haciendo click en "Submit". Al pinchar en cualquier punto de la ciudad, actualiza las coordenadas y recomendaciones al lugar pinchado.

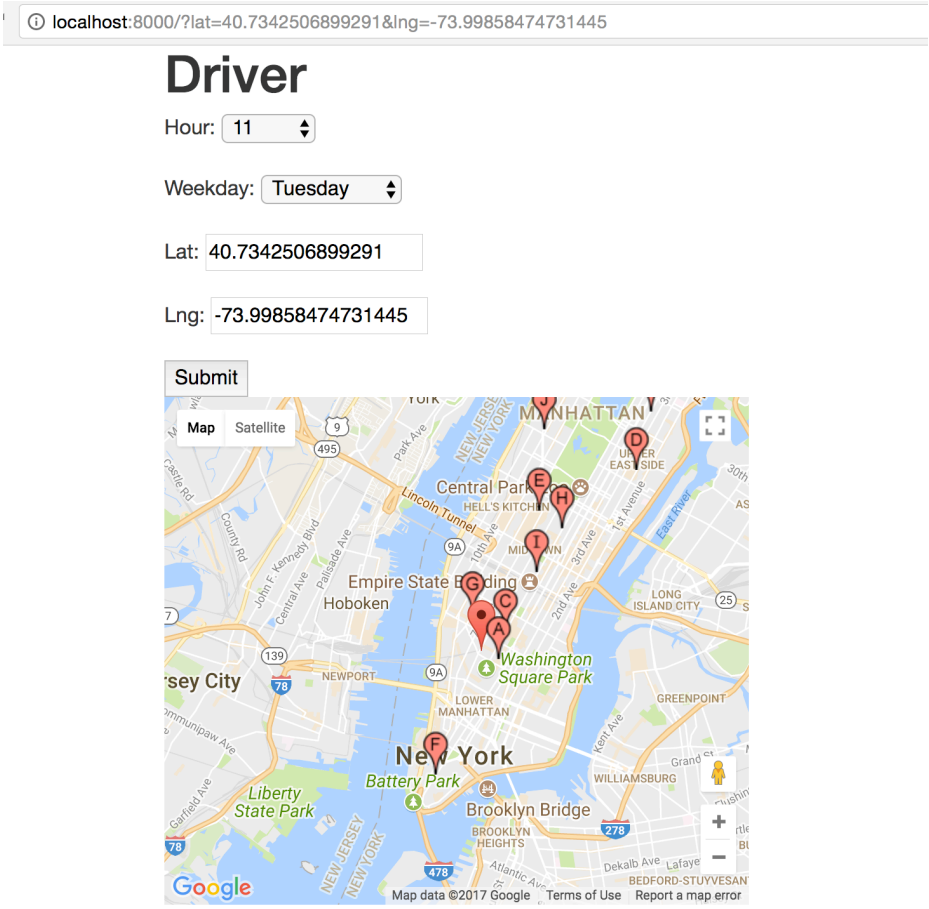


Figura 65. Página principal

Al pinchar en cualquiera de las recomendaciones del mapa, se abre Google Maps y se calcula la ruta desde la ubicación actual (Figura 66. Trayecto en Google Maps).

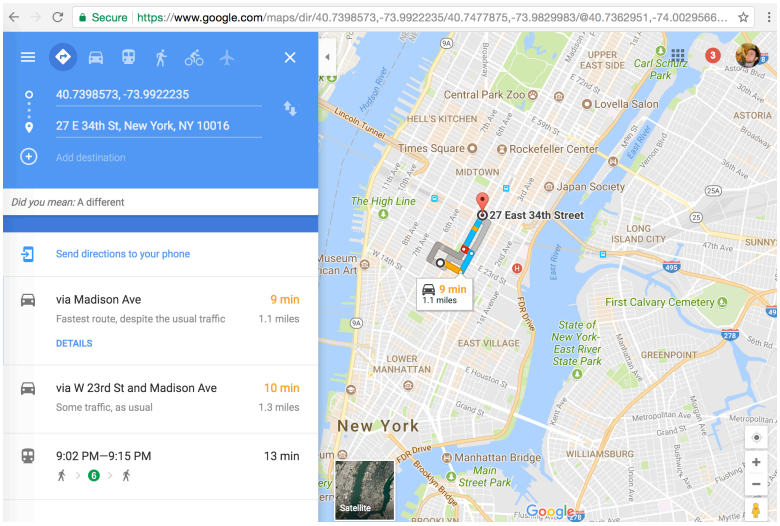


Figura 66. Trayecto en Google Maps

En el panel de administración se puede ver y ordenar un listado de las recomendaciones (Figura 67. Administración de recomendaciones: listado).

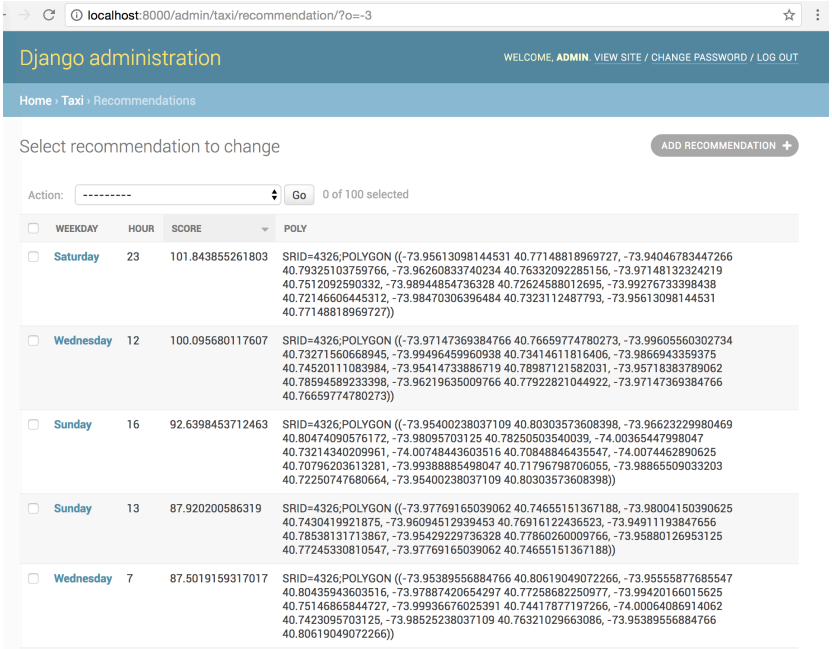


Figura 67. Administración de recomendaciones: listado

Al pinchar en cualquiera de las entradas, podemos modificar los parámetros de la recomendación, incluyendo los vértices del polígono Figura 68. Administración de recomendaciones: edición:

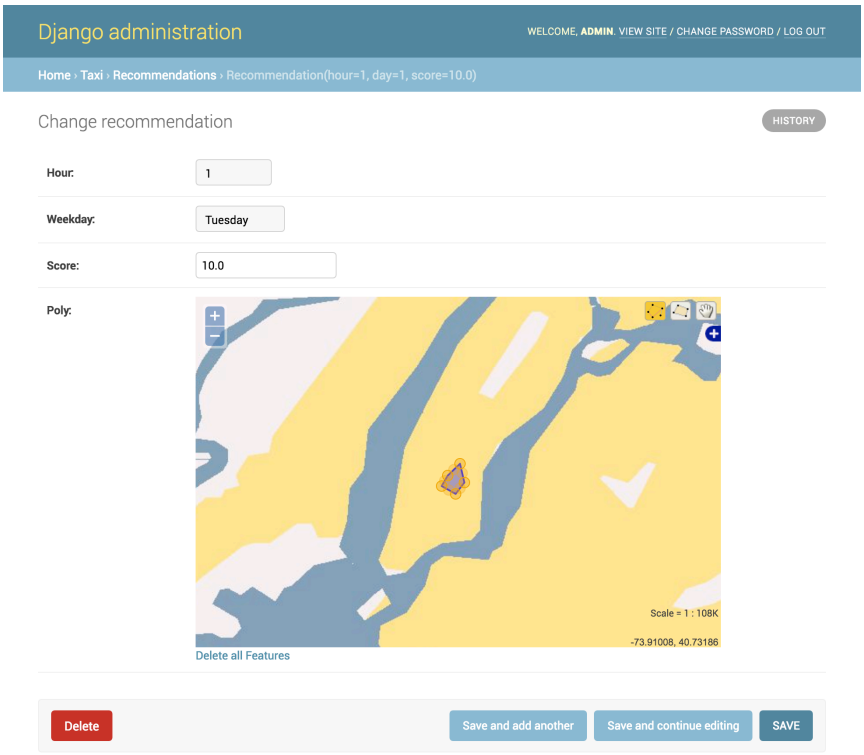


Figura 68. Administración de recomendaciones: edición

Anexo B: Propiedades completas de las trazas

VendorID	código que indica el proveedor TPEP que proporcionó el registro. 1 = Creative Mobile Technologies, LLC; 2 = VeriFone Inc.
tpep_pickup_datetime	Fecha y hora en que se activó el medidor.
tpep_dropoff_datetime	Fecha y hora en que se desconectó el medidor.
Passenger_count	Número de pasajeros en el vehículo.
Trip_distance	Distancia de viaje transcurrida en millas informada por el taxímetro.
Pickup_longitude	Longitud en la que el medidor estaba ocupado.
Pickup_latitude	Latitud en la que el medidor estaba ocupado.
RateCodeID	El código de tasa final en vigor al final del viaje. 1 = Tasa estándar 2 = JFK 3 = Newark 4 = Nassau o Westchester 5 = Precio negociado 6 = Paseo en grupo
Store_and_fwd_flag	Este indicador indica si el registro de viaje se mantuvo en la memoria del vehículo antes de enviar al vendedor, también conocido como "store and forward" porque el vehículo no tenía una conexión al servidor. Y = almacenar y reenviar viaje N = no almacenar y pasar un viaje hacia adelante
Fare_amount	La tarifa de tiempo y distancia calculada por el contador.
extra	Diversos extras y recargos. Actualmente, esto sólo incluye los $ 0.50 y $ 1 hora punta y cargos de nocturnidad.
MTA_tax	$ 0.50 El impuesto MTA que se activa automáticamente basado en el contador en uso.
Improvement_surcharge	$ 0,30 recargo de mejora de viajes evaluados en la bajada de la bandera.
Tip_amount	Propina. Sólo tarjeta de crédito.
Tolls_amount	Cantidad total de todos los peajes pagados en el viaje
Total_amount	La cantidad total cobrada a los pasajeros. No incluye consejos de efectivo.

Tabla 44. Propiedades completas de las trazas

Anexo C: Código

A continuación, se explican los diferentes módulos que se programaron para el desarrollo del proyecto.

Se puede consultar en https://github.com/jorgebg/taxi-recommendation-system.

`download_raw_data.sh`

Descarga los datos necesarios:

- Las trazas de los taxis
- Los bordes de los distritos en GeoJSON

`fabfile.py`

Gestiona el cluster Spark remotamente. Las acciones disponibles son:

- `config`
  - Configura el cluster. Admite parámetros para las diversas configuraciones utilizas en la sección Pruebas y Resultados.
- `cluster`
  - Inicia o para el cluster
- `info`
  - Muestra el modelo de procesador y la memoria total y libre.
- `notebook`
  - Inicia un notebook con Jupyter
- `pyspark`
  - Inicia una shell de pyspark
- `venv`
  - Instala las dependencias de Python usadas en el proyecto
- `ping`
  - Comprueba la latencia entre máquinas

`lib/`

Incluye las librerías compartidas por los diversos módulos del proyecto. Para su uso en el clúster Spark, se comprime en un archivo **`lib.zip`** que es compartido posteriormente entre todos los nodos.

`boros.py`

Lee el fichero GeoJSON con los bordes de los distritos de NYC y los transforma en polígonos para la librería Shapely de Python.

`plotly.py`

Configuraciones usadas recurrentemente en Plotly.

`mapbox.py`

Configuraciones usadas recurrentemente para la integración de Plotly en Mapbox.

`spark.py`

Configuración de Spark.

`process.py`

Filtrado y transformación del dataset.

`timer.py`
Librería de mediciones de tiempo.

`project/`
Contiene la aplicación web desarrollada en Django

`remote-shell.sh`
Comprime la librería `lib` para ser enviada a Spark y abre una shell pyspark.

`show_boros.py`
Muestra los bordes de los distritos para comprobar que cargamos el GeoJSON correctamente.

`show_process.py`
Muestra con precisión qué cantidad de registros se están filtrando por `lib/process.py`

`show_correlations.py`
Muestra las correlaciones entre pago total y distancia, tiempo y pasajeros.

`show_clusters.py`
Muestra que se pueden computar clusters para diferentes grupos de hora, y que cada resultado es distinto.

`test_gaussians.py`
Es el programa usado para ver la eficiencia de la computación del clusters con el Gaussian Mixture Model.

`compute_clusters.py`
Es el programa final que computa los clusters para cada hora del día y de la semana y guarda los resultados en `result.pickle`.

`result.pickle`
Los polígonos de los cluster computados junto con la hora del dia, de la semana, y la puntuación. Se serializan usando `pickle`, parte de la librería estándar de Python.

Anexo D: Normativa y Marco regulador

En relación al uso de las nuevas tecnologías, no existen inconvenientes legales directos. Sin embargo, pueden existir problemas de violación de la privacidad de las personas. De tal forma que estos problemas son tratados por los organismos correspondientes.

Toda aquella información que posibilite la identificación directa o indirectamente de cualquier persona se considera un dato de carácter personal. Por esto, el derecho fundamental a la protección de datos consiste en otorgar al ciudadano la capacidad de disponer, controlar y decidir sobre sus datos personales.[69]

A nivel nacional es la Agencia Española de Protección de datos es la autoridad de control independiente encargada de velar por el cumplimiento de la normativa que hace referencia a la protección de datos. Además, asegura y protege el derecho fundamental a la protección de datos personales.

La Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de carácter personal tiene como objeto garantizar y proteger todo lo relacionado con el tratado de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas así como el honor e intimidad personal y familiar[70]. Y los derechos que se engloban en esta ley son[71]:

- Derecho de información: Cuando se procede a la recogida de datos el interesado tiene que ser informado.
- Derecho de acceso: El interesado puede conocer y obtener de forma gratuita los datos de carácter personal que van a ser tratados.
- Derecho de rectificación: Se permite la corrección de errores o la modificación de datos que sean inexactos o incompletos.
- Derecho de cancelación: Se puede suprimir los datos considerados inadecuados o excesivos.
- Derecho de oposición: Derecho del afectado a que no puedan ser tratados sus datos personales.

En el ámbito europeo, se han fortalecido los derechos de los ciudadanos adaptando las reglas para los negocios a consecuencia de la en la era digital en la que nos encontramos. Se ha considerado que las empresas no pueden compartir datos de los usuarios sin una previa autorización en la que ofrecen su consentimiento al intercambio de datos. Y en el caso de violación de los derechos de privacidad de sus usuarios la multa a la que tendrán que enfrentarse las empresas puede ascender al 4% de los ingresos de la compañía.[72]

Referencias

[1] Yellow Cab, Long a Fixture of City Life, Is for Many a Thing of the Past (2017) URL: https://www.nytimes.com/2017/01/15/nyregion/yellow-cab-long-a-fixture-of-city-life-is-for-many-a-thing-of-the-past.html [19 de Septiembre del 2019]
[2] Flintrock, a command-line tool for launching Apache Spark clusters. URL: https://github.com/nchammas/flintrock [19 de Septiembre del 2019]
[3] TLC Trip Record Data, NYC Taxi & Limousine Commission. URL: http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml [19 de Septiembre del 2019]
[4] Nicholas Jing Yuan, Yu Zheng, Liuhang Zhang, Xing Xie. "T-Finder: A Recommender System for Finding Passengers and Vacant Taxis". 2010. URL: https://www.microsoft.com/en-us/research/publication/t-finder-a-recommender-system-for-finding-passengers-and-vacant-taxis/ [19 de Septiembre del 2019]
[5] Meng Qu Rutgers, Hengshu Zhu, Junming Liu, Guannan Liu, Hui Xiong. "A cost-effective recommender system for taxi drivers". 2014. URL: http://dl.acm.org/citation.cfm?id=2623668 [19 de Septiembre del 2019]
[6] HubCab. URL: http://hubcab.org/#12.00/40.7257/-73.8915 [19 de Septiembre del 2019]
[7] The Origins of 'Big Data': An Etymological Detective Story (Steve Lohr, 2013) URL: https://bits.blogs.nytimes.com/2013/02/01/the-origins-of-big-data-an-etymological-detective-story/ [19 de Septiembre del 2019]
[8] 10 Key Marketing Trends for 2017, IBM. URL: https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=WRL12345USEN [19 de Septiembre del 2019]
[9] Big Data, Investopedia. URL: http://www.investopedia.com/terms/b/big-data.asp [19 de Septiembre del 2019]
[10] Big Data, Wikipedia. URL: https://en.wikipedia.org/wiki/Big_data [19 de Septiembre del 2019]
[11]3D Data Management: Controlling Data Volume, Velocity and Variety (Laney, Douglas, 2001). URL: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf
[12] A Formal definition of Big Data based on its essential Features (De Mauro, Andrea; Greco, Marco; Grimaldi, Michele, 2016) URL: http://www.emeraldinsight.com/doi/abs/10.1108/LR-06-2015-0061 [19 de Septiembre del 2019]
[13] The 42 V's of Big Data and Data Science (Tom Shafer, 2017) https://www.elderresearch.com/company/blog/42-v-of-big-data [19 de Septiembre del 2019]
[14] Data Mining Curriculum: a Proposal. The Association for Computing Machinery's Special Interest Group on Knowledge Discovery and Data Mining (ACM KDD, 2006). http://www.kdd.org/curriculum/index.html [19 de Septiembre del 2019]
[15] New data-mining effort launched to study mental disorders (Robert Mitchum, 2011) URL: https://news.uchicago.edu/article/2011/10/10/new-data-mining-effort-launched-study-mental-disorders [19 de Septiembre del 2019]

[16] Distributed Network Systems: From Concepts to Implementations (Weijia Jia, Wanlei Zhou, 2005)
[17] Características Principales de los Sistemas Distribuidos, URL: http://sistemas-distribuidos-unerg.blogspot.ie/2008/10/caractersticas-principales-de-los.html [19 Septiembre de 2019]
[18] Distributed Aplicacion Arquitecture, Sun Microsystem. URL: http://java.sun.com/developer/Books/jdbc/ch07.pdf [19 de Septiembre del 2019]
[19] A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications, Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE (Rüdiger Schollmeier , 2002)
[20] What is a Grid? URL: http://dlib.cs.odu.edu/WhatIsTheGrid.pdf [19 de Septiembre del 2019]
[21] Cluster Computing: Applications (Bader, David, Mayo 2001). URL: http://www.cc.gatech.edu/~bader/papers/ijhpca.html [19 de Septiembre del 2019]
[22] IBM Sequoia worlds fastest computer (The Guardian, 2012) URL: https://www.theguardian.com/technology/2012/jun/18/ibm-sequoia-worlds-fastest-supercomputer [19 de Septiembre del 2019]
[23] MapReduce: Simplified Data Processing on Large Clusters (Jeffrey Dean, Sanjay Ghemawat, 2004)
[24] MapReduce, Wikipedia. URL: https://en.wikipedia.org/wiki/MapReduce [19 de Septiembre del 2019]
[25] Apache Spark, Wikipedia. URL: https://en.wikipedia.org/wiki/Apache_Spark [19 de Septiembre del 2019]
[26] Cluster Overview, Spark. URL: https://spark.apache.org/docs/latest/cluster-overview.html [19 de Septiembre del 2019]
[27] Apache Hadoop, Github URL: https://github.com/apache/spark [19 de Septiembre del 2019]
[28] Machine Learning Library Guide, Spark. URL: https://spark.apache.org/docs/latest/ml-guide.html [19 de Septiembre del 2019]
[29] Machine Learning, URL: https://www.sas.com/en_ie/insights/analytics/machine-learning.html [19 de Septiembre del 2019]
[30] Some Studies in Machine Learning Using the Game of Checkers (Samuel Arthur, 1959)
[31] Top-ranked Papers in "Data Mining", Microsoft Research. https://web.archive.org/web/20100421170848/http://academic.research.microsoft.com/CSDirectory/paper_category_7.htm [19 de Septiembre del 2019]
[32] K-means, Machine Learning Library, Spark. URL: https://spark.apache.org/docs/latest/ml-clustering.html#k-means [19 de Septiembre del 2019]
[33] k-means++: the advantages of careful seeding (Arthur, D.; Vassilvitskii, S., 2007)
[34] Scalable K-Means++ (Bahman Bahmani, 2012)
[35] A tutorial on clustering algoritms. URL: https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/mixture.html [19 de Septiembre del 2019]
[36] Maximum Likelihood from Incomplete Data via the EM Algorithm (Dempster, A.P.; Laird, N.M.; Rubin, D.B, 1977)
[37] Database Systems: Design, Implementation, & Management (Carlos Coronel, Steven Morris, 2016)
[38] Principles of transaction-oriented database recovery (Haerder, T., Reuter, A., 1983)

[39] A Relational Model of Data for Large Shared Data Banks (Codd, E.F., 1970)
[40] Spatial Databases, Encyclopedia of Computer Science and Engineering (Wiley, Cassie Craig, 2009)
[41] OGC History, http://www.opengeospatial.org/ogc/historylong/ [19 de Septiembre del 2019]
[42] Geospatial Queries, MongoDB, URL: https://docs.mongodb.com/manual/geospatial-queries/ [19 de Septiembre del 2019]
[43] The design of POSTGRES (Stonebraker, M; Rowe, LA, 1986)
[44] PostGIS, Wikipedia URL: https://en.wikipedia.org/wiki/PostGIS [19 de Septiembre del 2019]
[45] Feature List, PostGIS, URL: http://postgis.net/features/
[46] What is a Web Application? URL: https://www.lifewire.com/what-is-a-web-application-3486637 [19 de Septiembre del 2019]
[47] OSI Model, URL: https://en.wikipedia.org/wiki/OSI_model [19 de Septiembre del 2019]
[48] The OSI Model's Seven Layers Defined and Functions Explained. URL: https://support.microsoft.com/en-us/help/103884/the-osi-model-s-seven-layers-defined-and-functions-explained [19 de Septiembre del 2019]
[49] OSI Model, URL: https://en.wikipedia.org/wiki/OSI_model [19 de Septiembre del 2019]
[50] Front and back ends, Wikipedia. URL: https://en.wikipedia.org/wiki/Front_and_back_ends [19 de Septiembre del 2019]
[51] Google Maps JavaScript API Guide, URL: [19 de Septiembre del 2019] https://developers.google.com/maps/documentation/javascript/earthquakes
[52] Model-View-Controller, Wikipedia. URL: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller [19 de Septiembre del 2019]
[53] DigitalOcean – Growth, URL: http://trends.netcraft.com/www.digitalocean.com [19 de Septiembre del 2019]
[54] Scientific Computing Tools for Python. URL: https://scipy.org/about.html [19 de Septiembre del 2019]
[55] Spark Standalone Mode. URL: https://spark.apache.org/docs/latest/spark-standalone.html [19 de Septiembre del 2019]
[56] Design and Implementation of the Sun Network Filesystem (Russel Sandberg, 1985)
[57] Droplet, DigitalOcean. URL: https://www.digitalocean.com/products/compute/ [19 de Septiembre del 2019]
[58] SciPy. URL: https://www.scipy.org/ [19 de Septiembre del 2019]
[59] PlotLy. URL: https://plot.ly/ [19 de Septiembre del 2019]
[60] MapBox. URL: https://www.mapbox.com/ [19 de Septiembre del 2019]
[61] Top-ranked Papers in "Data Mining", Microsoft Research. https://web.archive.org/web/20100421170848/http://academic.research.microsoft.com/CSDirectory/paper_category_7.htm [19 de Septiembre del 2019]
[62] Borough Boundaries, CitiOFNewYork, URL: https://data.cityofnewyork.us/City-Government/Borough-Boundaries/tqmj-j8zm/data [19 de Septiembre del 2019]
[63] Shapely, Python library for geometric analysis, URL: https://pypi.python.org/pypi/Shapely [19 de Septiembre del 2019]
[64] Convex Hull, Scipy. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.ConvexHull.html [19 de Septiembre del 2019]

[65] pickle, Python Documentation, URL: https://docs.python.org/3/library/pickle.html [19 de Septiembre del 2019]
[66] Droplet, DigitalOcean. URL: https://www.digitalocean.com/products/compute/ [19 de Septiembre del 2019]
[67] Google Maps Developers, Google. URL: https://developers.google.com/maps/ [19 de Septiembre del 2019]
[68] GaussianMixture source code, Github. URL: https://github.com/apache/spark/blob/master/mllib/src/main/scala/org/apache/spark/mllib/clustering/GaussianMixture.scala [19 de Septiembre del 2019]
[69] Agencia española de protección de datos. Tu derecho fundamental a la protección de datos. URL: https://www.agpd.es/portalwebAGPD/CanalDelCiudadano/derechos/index-ides-idphp.php [19 de Septiembre del 2019]
[70] Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal. URL: http://noticias.juridicas.com/base_datos/Admin/lo15-1999.t1.html#t1 [19 de Septiembre del 2019]
[71] Agencia española de protección de datos. Principales derechos. URL: https://www.agpd.es/portalwebAGPD/CanalDelCiudadano/derechos/principales_derechos/index-ides-idphp.php [19 de Septiembre del 2019]
[72] El país. La UE aprueba la ley de protección de datos, bloqueada desde 2013. URL: http://internacional.elpais.com/internacional/2015/12/15/actualidad/1450208377_400556.html [19 de Septiembre del 2019]