El proceso de KDD en entornos de Big Data.

by

CRISTINA BERROCAL SAYAGO

A thesis submitted in conformity with the requirements
for the MSc in Economics, Finance and Computer Science

University of Huelva & International University of Andalusia

JULIO 2020

El proceso de KDD en entornos de Big Data

Cristina Berrocal Sayago

Máster en Economía, Finanzas y Computación

Antonio Peregrín Rubio

Universidad de Huelva y Universidad Internacional de Andalucía

2020

Abstract

The present final project aims at describing the process of obtaining knowledge from a big database whose magnitude is Big Data level. To carry it out, it will be detailed the complete procedure from a conceptual point of view, phase by phase including its different techniques and tools, starting from the Exploratory Data Analysis and then, the own Knowledge Discovery in Databases process. All this will be adapted to Big Data environments.

In order to put the entire process into practice, as well as to show the specific techniques employed, It will be illustrated by means of a simple example consisting of a set of data. Despite not knowing the magnitude of a set of data typical of Big Data environments, It will be treated as if It were applied in each case some Machine Learning techniques through the Apache Spark MLlib library.

JEL classification: C15, C8.

Keywords: KDD process, Exploratory Data Analysis, Big Data, Machine Learning, MapReduce, Apache Spark, Python.

Resumen

El propósito de este trabajo es describir el proceso de obtener conocimiento de un conjunto de datos grandes cuyo orden de magnitud es del nivel de Big Data. Para ello, se describirá de forma detallada el proceso completo desde un punto de vista conceptual, fase a fase con sus diferentes técnicas y herramientas, empezando por el Análisis Exploratorio de Datos y continuando posteriormente con el propio proceso de Knowledge Discovery in Databases, todo ello adaptado a los entornos Big Data.

Con objeto de mostrar todo el Para comprender mejor todo el proceso en la práctica, así como en concreto las técnicas empleadas, se ilustrará mediante un ejemplo sencillo consistente en un conjunto de datos que, pese a no reunir el tamaño de un conjunto de datos propio de entornos Big Data, se tratará como si lo fuera aplicando en cada caso técnicas de Machine Learning mediante la librería MLlib de Apache Spark.

Clasificación JEL: C15, C8.

Palabras clave: proceso de KDD, Análisis Exploratorio de Datos, Big Data, Machine Learning, MapReduce, Apache Spark, Python.

Agradecimientos

En primer lugar, me gustaría agradecer a mi familia el apoyo que me han brindado durante toda mi vida y mi etapa académica, a quienes les debo la mayor parte de mis logros.

Agradecer a mis abuelos los años en Huelva, ya que, sin ellos, no estaría aquí.

Agraceder también a mi pareja por apoyarme y aconsejarme en todo momento y por sus infinitas palabras de ánimo.

Agradecer a todo el profesorado de este máster su empeño y dedicación a la hora de transmitir los conocimientos y por hacer que aumente nuestro interés en aprender. En especial, gracias por el esfuerzo realizado durante los meses de confinamiento, y por permitirnos seguir con las clases con la mayor normalidad posible.

En concreto, agradecer a mi tutor, Antonio Peregrín Rubio, la infinita paciencia y dedicación prestadas durante la realización de este trabajo, que me ha servido para aprender e interesarme aún más por el mundo del Big Data.

Índice

Lista de Tablas

Lista de Figuras

Lista de Gráficos

1 Introducción

El increíble aumento del volumen de datos generados, junto con su necesidad de procesamiento, desde hace unas décadas hasta la actualidad, ha llevado a que surjan tecnologías específicas que se agrupan bajo el término Big Data para dar nombre a este fenómeno globalmente. Big Data es, por tanto, un concepto muy utilizado y que además está de moda, debido a la sociedad de la información en la que convivimos.

Asimismo, es un término conocido y empleado por la población en general para referirse a las técnicas que utilizan grandes empresas como Google, Facebook o Amazon para sacar conocimiento a partir de los datos de los que disponen. En realidad, son técnicas aplicadas a un amplio rango de campos del conocimiento, que irían desde la medicina, hasta las finanzas o la física. Los datos de estas áreas tan diversas tienen su origen en sensores, Internet, transacciones de todo tipo, e incluso los dispositivos móviles que nos acompañan en nuestra vida diaria.

Esta enorme cantidad datos de tipo muy diverso, muchas veces generados en tiempo real, resumen sus características en tres conceptos que generan una definición acertada del término Big Data. Estos conceptos son velocidad, volumen y variedad, de modo que podríamos definirlo como "conjuntos de datos que superan la capacidad del software habitual para ser capturados, gestionados y procesados en un tiempo razonable y por los medios habituales de procesamiento de la información" (Pérez, 2015, p. 5).

Actualmente se habla de Ciencia de los Datos para referirse de forma genérica a la disciplina que se ocupa del tratamiento de los datos mediante todo tipo de técnicas, tanto estadística como computacionales. El Knowledge Discovery in Databases (KDD) o Extracción de Conocimiento en Bases de datos es una disciplina que alcanzó hace ya algunos años su madurez y está compuesta por gran cantidad de métodos (inclusive Machine Learning, que son un conjunto de técnicas fundamentalmente de la Inteligencia Artificial Computacional) para obtener conocimiento oculto en conjunto de datos. La llegada posterior y más reciente de las necesidades de tratamiento de grandes volúmenes de datos y con ellas las tecnologías para hacerlo (Big Data), aportan a la actual Ciencia de los Datos la capacidad de extender el KDD y la propia estadística a los grandes volúmenes de datos.

El objetivo de este trabajo es agrupar y describir este proceso completo de KDD en entornos Big Data, es decir, describir quién es quién, cómo y para qué se utilizan las herramientas, e incluso ilustrarlo con un ejemplo, que, si bien no será un gran conjunto de datos (por limitaciones de recursos computacionales y tiempo), su tratamiento se realizará como si lo fuera.

2 Descripción del estado del arte

Desde hace décadas hasta la actualidad, ha resultado de gran interés extraer conocimiento de los datos que se generan por las empresas e instituciones, tanto públicas como privadas. Para ello, se ha hecho necesario estructurar la forma de proceder a la hora de extraer los datos en crudo y procesarlos hasta conseguir un resultado interpretable.

Este proceso estructurado e iterativo es el proceso de KDD o proceso de extracción de conocimiento de bases de datos. Resulta de gran importancia, ya que sin su aplicación, el ser humano sería incapaz de conocer mediante métodos convencionales (por ejemplo, los estadísticos o lo computacionales fuera del ámbito de la computación inteligente) el conocimiento, en el sentido de algo más que la información, que encierran estos datos.

Como se indicó previamente, la extensión de los métodos del KDD, que tan buenos resultados dan en conjuntos de dimensiones limitadas, a los grandes conjuntos de datos (bien por número de casos, ejemplos o instancias, sino también en número de atributos, variables o dimensión), se puede dar gracias a la llegada de la tecnología del Big Data, creada genéricamente para almacenar y procesar o transformar grandes conjuntos de datos.

El uso de estas tecnologías actuales que lo posibilitan serán el centro de este trabajo. Para ello, en esta Sección 2, se describirá por un lado, en qué consiste este proceso de KDD y, por otro se mostrarán las tecnologías para Big Data que permiten el manejo de estos datos.

2.1 El proceso de KDD

Este proceso ordenado busca identificar patrones y relaciones válidas, novedosas, útiles y comprensibles a partir de grandes bases de datos complejas mediante análisis y modelos exploratorios automáticos, de acuerdo con la definición dada por Maimon & Rokach (2010). Su aplicación es fundamental debido a que el valor de los datos está en la información y, posteriormente, en el conocimiento, que se puede extraer de ellos.

Es un proceso secuencial y muchas veces iterativo. Se inicia tras la delimitación de los objetivos que se quieren alcanzar con él y que determinarán algunas de sus fases, de acuerdo con Maimon & Rokach (2010).

2.1.1 Fases del proceso de KDD

Una vez se han definido y se conocen mejor los datos necesarios, se inician las fases que componen el proceso de KDD.

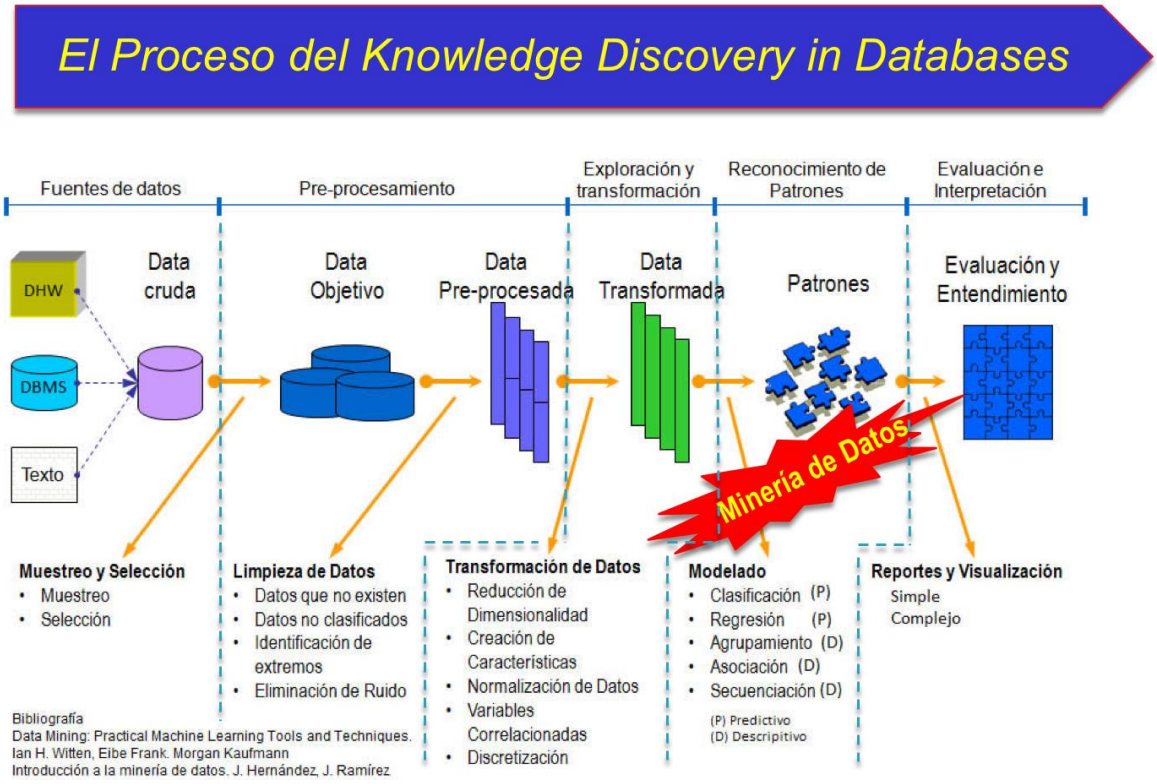

Figura 2.1. Fases del proceso de Knowledge Discovery in Databases (Fases del Proceso de Knowledge Discovery in Databases, s.f.).

El preprocesamiento de los datos es la fase que abre el proceso. Se trata de un proceso clave para que los algoritmos que se apliquen en las fases posteriores funcionen correctamente y se optimicen en tiempo de cómputo, e incluso en consumo de energía de los equipos informáticos, que es un elemento de sostenibilidad no despreciable.

En primer lugar, se transforman los datos extraídos del mundo real en un formato legible por la máquina. Luego, se aplican técnicas de limpieza, imputación, eliminación de inconsistencia y

duplicidad y reducción de los datos. Al trabajar con datos en crudo, los que vienen del mundo real, si este paso no se realiza, surgen una serie de problemas que llevan a resultados pobres (García et al., 2016).

En esta fase se distinguen técnicas que buscan la reducción de los datos, técnicas que permiten el tratamiento de datos imperfectos y técnicas que permiten corregir los conjuntos de datos no balanceados. Todas ellas se describen a continuación.

- *Técnicas de reducción de los datos*, se componen de algoritmos de preprocesamiento empleados para simplificar y limpiar los datos. Logran que los algoritmos de Minería de Datos (aplicados en la siguiente fase) sean más rápidos y que tengan una mayor precisión en su ejecución. Para ello, emplean tres tipos de técnicas según la naturaleza de los elementos que reduzcan: la reducción de la dimensionalidad, la reducción de instancias y la discretización, de acuerdo con Pyle (1999) (Citado en Luengo et al., 2020). A continuación se pasa a describirlas.

    - *Reducción de dimensionalidad*, su aplicación es muy necesaria. Conforme aumenta el número de variables independientes respecto al número de instancias del conjunto de datos, el riesgo de incurrir en la maldición de la dimensionalidad aumenta en los algoritmos de Minería de Datos posteriores.

      Esta maldición de la dimensionalidad conlleva que las variables explicativas pierdan poder de predicción. Al aumentar el número de estas variables, será más complicado diferenciar entre las características de los datos que recoge cada una de ellas. Este hecho destaca la importancia de aplicar este tipo de técnicas.

    - *Selección de atributos*, es una técnica que busca limpiar los datos mediante la identificación y eliminación de todos aquellos datos que no son importantes o que redundan en el conjunto de datos, de acuerdo con Hall (1999) (citado en Luengo et al. 2020). Obtiene un conjunto de datos menor, o subconjunto de datos, con aquellos atributos que verdaderamente son relevantes y describen los datos (Luengo et al., 2020).

Esta técnica elimina la posible correlación que pueda existir entre atributos, reduce el riesgo de sobreajuste de los algoritmos de Machine Learning y reduce el espacio de características. Por lo que ayuda también a reducir el tiempo de ejecución del algoritmo aplicado más adelante y la memoria empleada para ello, (Luengo et al., 2020).

- *Extracción de atributos*, estas técnicas generan un nuevo conjunto de datos con atributos generados de forma artificial. Estos atributos son la combinación de atributos originales mediante la aplicación de un análisis factorial o un Análisis de Componentes Principales, según Jolliffe (2011) (citado en Luengo et al., 2020).

- *Reducción de instancias*, son técnicas complementarias a la extracción de atributos que buscan reducir el tamaño del conjunto de datos eliminando instancias o generando otras nuevas. Todo ello sin que la calidad del conocimiento extraíble de él se vea reducida (Luengo et al., 2020).

- *Selección de instancias*, se trata de un conjunto técnicas muy necesarias, de acuerdo con Liu & Motoda (2002) (citado en Luengo et al., 2020). Son capaces de seleccionar un subconjunto de datos que reemplazan al original mediante operaciones inteligentes de categorización de instancias. Además de alcanzar el objetivo de la fase de Minería de Datos a partir de este subconjunto, (Luengo et al., 2020).

  También permiten limpiar los datos mediante la eliminación de instancias ruido e instancias redundantes. De este modo, permiten que los algoritmos aplicados en la fase de Minería de Datos puedan trabajar en entornos Big Data y enfocarse en las instancias verdaderamente importantes, (Luengo et al., 2020).

- *Generación de instancias,* son una serie de técnicas que generan y reemplazan a los datos originales con datos generados de forma artificial. Estos datos servirán para completar regiones vacías del dominio del problema y para reducir el número de instancias del conjunto de datos. Esto último lo consigue condensando las instancias en un número de ellas, (Luengo et al., 2020).

- *Discretización,* es una de las técnicas más empleadas en el preprocesamiento de los datos. Consiste en transformar un gran conjunto de datos que contiene valores

numéricos en un conjunto de datos menor, que contenga valores categóricos, de acuerdo con Luengo et al. (2020).

- *Datos imperfectos*, los algoritmos de Minería de Datos deben trabajar, en teoría, sobre un conjunto de datos sin ruido. En la práctica, los datos de la vida real no aseguran este hecho. Por este motivo, se hace necesario eliminar este tipo de datos o imputarlos, de acuerdo con García-Gil et al. (2002) (citado en Luengo et al., 2020).

    - *Missing Values* o valores perdidos, es muy común encontrar valores de este tipo en los datos que provienen de la vida real. Se trata de datos que aparecen vacíos debido a errores o limitaciones en el proceso de medición o de muestreo o a restricciones presupuestarias. Deben tratarse con cuidado debido a que pueden generar problemas en la fase de Minería de Datos, (Luengo et al., 2020).

      Se suelen generar debido a errores humanos o en los sensores. De forma que se clasifican en dos tipos: datos ausentes prescindibles y no prescindibles (Salvador Figueras & Gallardo, 2003).

      Los primeros son los generados por errores humanos y el usuario puede controlarlos e identificarlos de forma explícita. Se identifican a su vez con los llamados datos censurados u observaciones incompletas (Salvador Figueras & Gallardo, 2003). Surgen al trabajar con muestras (no se dispone de toda la información de la población) o por efectos de calendario (al finalizar un trimestre, por ejemplo, aún no se dispone de los datos para este periodo).

      Los segundos surgen debido a errores en sensores y otros procesos o herramientas. No están bajo el control del usuario y no se pueden identificar de forma explícita, Salvador Figueras & Gallardo (2003).

      Para solucionar el problema que generan, se emplean técnicas de imputación a partir de su media o su moda. También existen técnicas más sofisticadas que se valen de Machine Learning para obtener patrones presentes entre el resto de valores y predecir así los valores perdidos.

- *Tratamiento del ruido*, se trata de valores que generan alguna perturbación en el conjunto de datos. Las técnicas de Minería de Datos aplicadas en la siguiente fase de este proceso de KDD serán incapaces de generar resultados adecuados si no se tratan estos valores.

  El tratamiento se basa en métodos para pulir los datos y en métodos de filtrado del ruido. Los primeros se emplean cuando el ruido afecta al etiquetado de las instancias. Los segundos, por otro lado, identifican y eliminan estos datos problemáticos del conjunto de entrenamiento y no conllevan que se modifique la técnica de Minería de Datos aplicada en la siguiente fase del proceso, (Luengo et al., 2020).

- *Conjuntos de datos no balanceados*, se trata de un problema relacionado con la probabilidad de que una determinada instancia pertenezca a una determinada clase. Este problema se conoce como problema del desequilibrio de clase y se suele dar en problemas de clasificación. Se trata de solucionar aplicando técnicas de preprocesamiento que remuestrean los datos para balancear el número de instancias que pertenecen a cada clase, (Luengo et al., 2020).

  Dentro de estas técnicas se pueden encontrar métodos de submuestreo, que eliminan instancias mayoritarias y crean un nuevo subconjunto a partir de estas instancias. El segundo grupo de métodos engloba los métodos de sobremuestreo, que replican instancias del conjunto de datos original o crean instancias nuevas a partir de estas, (Luengo et al., 2020). La técnica más empleada es la técnica de sobremuestreo de minorías sintéticas, que interpola instancias de aquellas clases minoritarias que se encuentran juntas (por ejemplo, el algoritmo SMOTE), de acuerdo con García (2016) (citado en Luengo et al., 2020).

  Una vez se haya obtenido el conjunto de datos limpio de todo este tipo de valores que pueden generar problemas en la implementación de los algoritmos. Se pasa a la siguiente fase, la de Minería de Datos.

La segunda fase del proceso es la de Minería de Datos propiamente (aunque a veces, a todo el proceso de KDD se le identifica como Minería de Datos). Lo primero será identificar la técnica más adecuada para afrontar el problema a resolver. Para ello, se debe considerar el tipo de datos

disponible y el objetivo que se quiere conseguir. Más tarde, se adaptarán estas técnicas a los datos de los que se dispone para poder implementarlas y generar resultados.

Puede definirse como "el núcleo del proceso de KDD, engloba la inferencia de algoritmos que exploran los datos, desarrollan el modelo y descubren patrones previamente desconocidos. El modelo se emplea para entender fenómenos en los datos, analizar y predecir" (Maimon & Rokach, 2010, p. 22).

Como disciplina, combina el Aprendizaje Automático o Machine Learning y la Estadística y hace hincapié en "la escalabilidad del número de atributos y de instancias, en los algoritmos y arquitecturas (...) y la automatización para manejar grandes volúmenes de datos heterogéneos" (Riquelme et al., 2006, p. 13). Agrupándose las técnicas que la componen en las siguientes áreas:

- *Aprendizaje supervisado,* se compone de un conjunto de problemas en los que la variable de salida está perfectamente definida. Su finalidad es predecir el valor de esta variable para nuevas muestras de datos desconocidos. Para ello, aprende las relaciones existentes entre las variables de entrada y esta variable mediante dos técnicas:

    - *Regresión:* el dominio de la variable de salida es continuo.

    - *Clasificación*: la variable de salida toma valores en un dominio discreto y los valores que puede tomar son conocidos. Estos valores se suelen denominar etiquetas o clases.

- *Aprendizaje no supervisado,* a diferencia de los problemas anteriores, en este área los valores de la variable de salida son desconocidos. Por lo que buscan identificar relaciones descriptivas implícitas en los datos de dos formas:

    - Asociación: detecta relaciones o patrones entre las variables, como la correlación.

    - Clustering: identifica grupos de instancias con características similares, maximizando la distancia entre los distintos grupos y la cohesión entre las instancias pertenecientes a un mismo grupo.

      Dentro de las técnicas de clustering encontramos dos tipos, *soft clustering* y *hard clustering.* De acuerdo con Withanawasam (2015), las primeras permiten que una

instancia pertenezca a más de un clúster, como es el caso del algoritmo *Fuzzy K-Means*. A diferencia de las segundas, también conocidas como *clustering* exclusivo. Un ejemplo de ellas es el algoritmo *K-Means.*

- *Aprendizaje semi-supervisado*: en este tipo de problemas la variable de salida sólo está definida para un conjunto minoritario de los datos. Valiéndose de las técnicas anteriores de aprendizaje supervisado y no supervisado para extraer conocimiento valioso de los datos.

Siguiendo a Maimon & Rokach (2010), otra forma de clasificar las técnicas de Minería de Datos consiste en diferenciar técnicas basadas en el descubrimiento y técnicas orientadas a la verificación.

Las técnicas basadas en el descubrimiento buscan identificar de forma autónoma nuevas reglas y patrones mediante métodos de predicción y en métodos descriptivos, de acuerdo con Maimon & Rokach (2010).

De acuerdo con estos autores (Maimon & Rokach, 2010), la finalidad de los métodos de predicción es construir de forma automática un modelo de comportamiento que obtenga nuevas muestras desconocidas y sea capaz de predecir los valores de las variables relacionadas con ella. “También desarrollan patrones que reúnen el conocimiento extraído de ellos, de forma que sean comprensibles y fáciles de operar” (Maimon & Rokach, 2010, p. 5). Mientras que los métodos descriptivos están orientados a interpretar los datos, entender cómo se distribuyen, por ejemplo.

Según Maimon & Rokach (2010), una gran parte de las técnicas de descubrimiento se fundamentan en el método inductivo. Construyen un modelo de forma implícita o explícita, generalizando a partir de un número suficiente de instancias de entrenamiento. Asumen, para ello, que el modelo entrenado puede generalizarse a futuros datos desconocidos.

A diferencia de los métodos anteriores, las técnicas de verificación evalúan hipótesis realizadas por el usuario. Se fundamentan en las técnicas estadísticas tradicionales, como contrastes de hipótesis (Maimon & Rokach, 2010).

Pueden estar menos relacionados con la Minería de Datos, debido a que la mayoría de problemas de esta rama buscan descubrir nuevas hipótesis y a que el enfoque estadístico se basa en la estimación de un modelo. Mientras que la Minería de Datos, busca identificar el modelo y

construirlo a partir de una evidencia, de acuerdo con las ideas expuestas por Maimon & Rokach, (2010).

Hay que añadir que en la práctica encontramos dos subfases en la etapa de Minería de Datos. En la primera, se entrena el modelo elegido de entre los anteriores, de forma que pueda reconocer las relaciones o patrones existentes entre las variables para datos conocidos. Una vez se ha validado la bondad del ajuste de esta subfase de entrenamiento, se puede proceder a validar el modelo en su conjunto empleando datos desconocidos. Se inicia aquí la fase de prueba del modelo.

Si el modelo no es capaz de generar un buen ajuste para los datos de entrenamiento, se deberán observar los posibles errores que se estén cometiendo y volver atrás hasta el punto en el que se puedan solventar. Una vez se haya hecho esto, se entrenará de nuevo el algoritmo.

Por otro lado, se debe tener en cuenta que los algoritmos aplicados para extraer conocimiento deben ser certificables. Es decir, el usuario de los resultados extraídos del proceso debe poder entender el conocimiento tras ellos, si no, no tendrá sentido haber invertido tiempo y esfuerzo en este proceso.

Por ejemplo, muchas veces una red neuronal puede generar mejores resultados que algunas técnicas menos sofisticadas de Minería de Datos. El problema es que el ser humano puede no ser capaz de entender por qué y cómo han generado esos resultados. Muchas veces se basan en un sistema de caja negra y no se sabe qué ocurre entre las capas de la red, por lo que puede ser más adecuado seleccionar una técnica menos sofisticada pero que el usuario sea capaz de entender completamente.

Finalmente, la fase de interpretación de los datos es la que finaliza el proceso de KDD. En ella se extrae conocimiento de valor y útil para los usuarios a partir de los resultados que se desprenden de la aplicación de las técnicas anteriores. En esta fase también deberemos validar estos resultados obtenidos, ya que, de acuerdo con Maimon & Rokach (2010), el éxito de esta fase determinará la efectividad de todo el proceso de KDD.

2.1.2 Análisis Exploratorio de Datos

El Análisis Exploratorio de los Datos o Exploratory Data Analysis (EDA) hace referencia a una forma de analizar los datos definida en 1977 por John W. Tukey en su libro *Exploratory Data*

Analysis. Emplea técnicas de Estadística Descriptiva para estudiar los principales estadísticos básicos como son la media, la mediana, la moda y las desviaciones típicas. Asimismo, emplea técnicas de representación de los datos para observar los datos mediante gráficas de distribución o histogramas, entre otras, y aplica inferencia estadística para validar la muestra. Con todo esto, se obtiene el primer acercamiento a los datos.

Según NIST Sematech (2006) (citado en Chon Ho, 2010) las distintas técnicas de Análisis Exploratorio de Datos buscan maximizar la comprensión de los datos, descubrir estructuras implícitas, extraer las variables más importantes, detectar valores atípicos y anomalías en los datos, probar supuestos subyacentes, desarrollar modelos y determinar el factor óptimo del ajuste del modelo.

Siguiendo a Salvador Figueras & Gallardo (2003), este análisis comienza con la preparación de los datos para calcular los estadísticos básicos (moda, mediana, media, desviación típica, entre otras). A continuación, se realiza un análisis gráfico para conocer la naturaleza de los atributos que componen el conjunto de datos y si existe algún tipo de relación entre ellos.

En este sentido, siguiendo a Becher et al. (2000), el Análisis Exploratorio de los Datos busca descubrir atributos inapropiados para la aplicación de técnicas de Minería de Datos y atributos sospechosos. Los primeros se eliminan del conjunto de datos de forma automática y pueden ser del siguiente tipo:

- *Atributos constantes:* aquellos que contienen un único valor en todo su dominio (Becher et al., 2000).

- *Atributos nulos:* no toman ningún valor en su dominio, sólo contienen valores nulos o perdidos (Becher et al., 2000).

- *Atributos casi nulos:* toman como valor una fracción de un valor nulo o perdido mayor a un determinado límite (Becher et al., 2000).

- *Atributos con más de un valor:* como su nombre indica, contienen más de un valor o una fracción mayor que un determinado límite. Son los relacionados con números de teléfono, DNI, etc. (Becher et al., 2000).

Mientras que los segundos, siguiendo a estos autores (Becher et al., 2000), generan cierto escepticismo a la hora de eliminarlos o no. Este análisis permite identificar los siguientes tipos de atributos sospechosos:

- *Artefactos:* guardan una determinada asociación o correlación con el objetivo, mayor a un valor de referencia. "Suelen incluirse de forma no intencionada y, si no se identifican correctamente, conducen a modelos que son artificialmente buenos pero no generalizan bien" (Becher et al., 200, p. 425).

- *Predictores pobres:* según Becher et al. (2000), al igual que los anteriores, son atributos que guardan una determinada asociación o correlación con el objetivo pero, en este caso, es inferior a un cierto valor de referencia. Como su nombre indican, no tienen mucho poder de predicción sobre el resultado que se quiere obtener, pero si se combinan adecuadamente con otros atributos, puede incrementarse su poder predictivo.

- *Atributos con valores casi constantes:* un valor del atributo representa la mayoría de los valores posibles que puede tomar ese atributo en su dominio (Becher et al., 2000).

- *Atributos con pocos valores:* son aquellos que no toman suficientes valores para las instancias que se poseen (Becher et al., 2000).

- *Atributos con pocas instancias:* son aquellos que toman un número de valores no nulos muy escasos (Becher et al., 2000).

Una vez se hayan identificado, se debe decidir qué se hace, según su naturaleza, con cada tipo de estos atributos y se debe evaluar su impacto sobre las técnicas que se aplicarán más tarde a los datos.

Además de estos atributos, siguiendo a Salvador Figueras & Gallardo (2003), existen valores que pueden generar problemas. Son los denominados *outliers* o valores atípicos y pueden generarse por distintos motivos, como son errores de procedimiento o de codificación, acontecimientos extraordinarios, variables observadas únicamente para una combinación de valores de las variables o instancias extraordinarias para las que el usuario no encuentra explicación.

De acuerdo con estos autores (Salvador Figueras & Gallardo, 2003), los valores atípicos generados por errores de procedimiento o de codificación se pueden filtrar o recodificar como valores ausentes (que se tratarán en la fase de preprocesamiento del proceso de KDD). Si esto es imposible, se pueden eliminar, al igual que se hace con los valores atípicos generados por acontecimientos extraordinarios.

Según Salvador Figueras & Gallardo (2003), las variables observadas únicamente para una combinación de valores de las variables se deben estudiar para conocer su influencia sobre las fases posteriores del análisis. Mientras que, en el caso de las instancias extraordinarias, se pueden realizar las fases posteriores con y sin estas instancias para comprobar su influencia sobre el análisis. En caso de existir esta influencia, se deberá estudiar por qué existen estas instancias y exponer este hecho en las conclusiones del análisis.

Estos valores se pueden identificar valores mediante histogramas, gráficos de dispersión, gráficos de caja o el cálculo de puntuaciones tipificadas, de acuerdo con estos autores (Salvador Figueras & Gallardo, 2003).

En este análisis, además de identificar todos estos tipos de atributos que presentan o generan algún problema en el desarrollo del proceso de Extracción de Conocimiento. Se deben identificar los distintos tipos de atributos con los que se trabaja según el tipo de valores que almacenan.

Las variables o atributos cualitativos son aquellas que toman valores no numéricos o de tipo cadena y que se refieren a categorías o características de las instancias, de acuerdo con Salvador Figueras y Gargallo (2003). Dentro de este tipo de variables se pueden distinguir tres tipos de variables: nominales, ordinales y binarias.

Las variables nominales son aquellas que recogen grupos de valores asociados a los datos y a cada valor se le puede asignar un número. Sin que pueda establecerse un orden entre ellos o se le pueda dar un significado. Un ejemplo de este tipo de variables puede ser un atributo que recoja distintos tipos de transporte para realizar un viaje (a cada transporte se le da un valor numérico, sin que sea uno mejor que otro o pueda realizar un ranking).

Las variables ordinales, al igual que las anteriores, recogen grupos de valores asignados a los datos. Pero, en este caso, los valores numéricos asociados a ellos sí tienen un significado y se puede

establecer un orden entre ellos. Un ejemplo de estas variables es la escala de Likert, muy utilizada cuando se diseñan cuestionarios.

Finalmente, las variables binarias recogen mediante los valores numéricos 0 y 1 dos posibles respuestas expresadas en los datos. En realidad, es un tipo de variable nominal que sólo contiene dos categorías de instancias. Un ejemplo de estas variables es la codificación, mediante estos valores 0 y 1, de que los individuos de una muestra tengan hijos o no. El 0 representaría que el individuo no tiene hijos y el 1, que sí los tiene.

Las variables o atributos cuantitativos son aquellas cuyos valores son de tipo numérico. Pueden ser atributos discretos, si sólo toman valores enteros, o atributos continuos, si toman o pueden tomar valores con cifras decimales. Para representar estos atributos se emplean las mismas formas de representación que en el caso de las variables cuantitativas si se agrupan los valores por intervalos (Salvador Figueras y Gargallo, 2003).

En ambos casos, se pueden y se deben realizar tablas de frecuencias para construir estas gráficas. Esto también ayuda a observar los valores que más se repiten y los que menos y a calcular estadísticos básicos como la media, la moda o la desviación típica de cada atributo.

Siguiendo con las fases definidas por Salvador Figueras & Gallardo (2003), el Análisis Exploratorio de Datos continúa con un análisis descriptivo de tipo numérico para cuantificar aspectos gráficos de los atributos, como la correlación entre ellos. Para comprobar esto último se construye la matriz de correlación entre variables.

Tras esto, se evalúan supuestos básicos como la distribución que siguen los datos, la linealidad y la homocedasticidad. Se identifican los atributos que puedan generar algún tipo de problema y se evalúa su impacto potencial a la hora de aplicar los análisis estadísticos necesarios (Salvador Figueras & Gallardo, 2003).

En este sentido, el estudio de la distribución se puede realizar una vez se conocen los datos a través de sus estadísticos. Se pueden emplear para ello métodos gráficos y contrastes de hipótesis.

Los métodos gráficos engloban técnicas de representación como los histogramas, que ayudan a comparar la distribución que siguen los datos con la forma de una determinada distribución, como puede ser la Normal. También se pueden emplear diagramas de cuantiles, que comparan los

cuantiles muestrales con los esperados bajo la hipótesis de que los datos siguen una determinada distribución, Salvador Figueras & Gallardo (2003).

Por otro lado, los contrastes de hipótesis se basan en el cálculo de determinados estadísticos de contraste y en la comparación con un cierto valor de referencia para determinar si se encuentra o no dentro de la región de rechazo del test. Se formula también una hipótesis nula (H0) sobre la distribución de los datos que se querrá aceptar o rechazar para un determinado valor de confianza o alpha (α), que suele tomar valores entre el 0.01 y el 0.05. Frente a esta hipótesis nula, se formulará otra contraria a ella, llamada hipótesis alternativa (H1 o Ha).

Salvador Figueras & Gallardo (2003) ilustran dos contrastes para el caso de testar si la distribución seguida por los datos es una Normal. Estos contrastes son el test de Kolmogorov-Smirnov y el test de Shapiro-Wilks.

Otra de las hipótesis que se deberá comprobar es la de homocedasticidad, es decir, que la varianza de los errores es constante a largo plazo. Para ello, se emplean las técnicas estadísticas de Análisis de la Varianza, Análisis Discriminante y Análisis de Regresión. Un test muy utilizado, en este sentido, el test de Levene, cuya hipótesis nula es la homocedasticidad y la alternativa, la heterocedasticidad (Salvador Figueras & Gallardo, 2003).

El Análisis Exploratorio de Datos concluirá cuando se conozcan perfectamente los datos con los que se trabaja.

2.2 Tecnologías para Big Data

La escalabilidad es uno de los problemas que surgen al tratar grandes volúmenes de datos con una gran variedad de tipos o complejidad. Consiste en la capacidad de hacer que un sistema crezca de forma sencilla ampliando el número de elementos de proceso, pero sin modificar los algoritmos ni sus parámetros, de forma transparente. Para conseguirlo, se vuelve obligatorio emplear paradigmas de programación distribuida como MapReduce. “Un modelo de programación y una implementación asociada para procesar y generar grandes conjuntos de datos” (Dean & Ghemawat 2008, p. 1).

El paradigma MapReduce se construye sobre dos funciones básicas que le dan nombre: *map* y *reduce*. La función *map* permite al usuario especificar una función que procesa una pareja de

clave/valor. La función *reduce*, le posibilita crear una función que agrupa todos los valores de una misma clave generados en la función *map*. Según la descripción de estos dos funciones dada por Dean & Ghemawat (2008).

Entre las funciones *map* y *reduce* existe una función intermedia automática llamada *shuffle* o *sort and shuffle*. Esta se encarga de reordenar las salidas de la función *map*, agrupando cada clave con sus valores asociados. Se puede observar el flujo de procesos MapReduce en la siguiente figura.

Figura 2.2. Flujo de procesos MapReduce (Hernández Yeja, 2015)

Según exponen Dean & Ghemawat (2008), la primera versión de este paradigma de programación creado por Google fue escrita en el año 2003. Estos autores atribuyen el éxito de su aplicación a distintos propósitos en esta compañía a su facilidad de uso. Procesos como la parelización o la tolerancia a fallos se ejecutan de forma transparente para el usuario o programador. Además, es capaz de resolver una gran variedad de problemas, ejemplo de ello es que la empresa lo usa para resolver el 80% de sus procesos. Permite también un uso eficiente de las máquinas que componen el clúster y añadir nuevas máquinas de una buena relación calidad precio.

De forma más detallada, MapReduce se basa "en iterar sobre los inputs, computar los pares clave/valor para cada trozo de entrada, agruparlos en valores intermedios por claves, iterar sobre los grupos resultantes y reducir estos grupos" (Lämmel, 2007, p. 1). Siguiendo con Lämmel (2007), se trata de un modelo muy simple que soporta de manera efectiva la paralelización del tratamiento de los datos. Permite al usuario o programador no tener que preocuparse de la programación distribuida y paralelizada; MapReduce se encarga de todo esto. Para ello emplea clústers de ordenadores, cada uno de ellos compuesto de nodos u ordenadores esclavos encargados de realizar una parte del trabajo encargado por el nodo maestro. Todos estos nodos esclavos deben

estar conectados al nodo maestro para reportarle los resultados de su trabajo y que pueden seguir trabajando.

De este modo, de acuerdo con Dean & Ghemawat (2008), MapReduce es tolerante a fallos, el nodo maestro es capaz de monitorear en todo momento el estado de los nodos esclavos. Si uno o varios de ellos falla, es capaz de asignar el trabajo a otro nodo que almacena los mismos datos que él o los nodos que han caído. Además, el nodo maestro escribe periódicamente copias de seguridad de su trabajo para que, en caso de que caiga, poder iniciar fácilmente la última copia.

Siguiendo a Rathi y Lohiya (2014), la base del entorno y el desempeño de MapReduce se compone de unas características propias de este modelo de programación y de una arquitectura.

Estas características propias de MapReduce expuestas por Schenkel (s.f.) (citado por Rathi y Lohiya, 2014) lo definen como una solución completa para la programación distribuida, un modelo con una interfaz simple y, a la vez, potente, con implementaciones generadas en horas y capaz de detectar fallos en la máquina y redistribuir el trabajo (tolerancia a fallos), además de evitar las pérdidas de datos debidas a fallos en el disco duro.

Por otro lado, según los autores Rathi y Lohiya (2014), su arquitectura se caracteriza por un proceso maestro específico para identificar procesos o máquinas esclavos para realizar las funciones *map* y *reduce*. Este mismo nodo maestro particiona el archivo de entrada que contiene los datos en n partes y los asigna a los nodos esclavos.

Los nodos esclavos encargados de la función *map* devuelven m archivos, los almacenan en el disco duro y lo notifican al nodo máster. Por otro lado, cada nodo esclavo encargado de la tarea reduce, lee cada archivo devuelto por los nodos esclavos encargados de la función *map* y los ordenan, agregando datos por cada clave. Según lo expuesto por estos autores (Rathi y Lohiya, 2014)

2.2.1 Apache Hadoop

Hadoop fue creado en el año 2005 por Doug Cutting y Mike Cafarella de la compañía Yahoo! como alternativa de código abierto a MapReduce de Google. En 2008 se convirtió en un proyecto de Apache y, desde entonces, es un software open source de Apache.

“Permite procesar de forma distribuida grandes conjuntos de datos mediante clúster de ordenadores empleando modelos de programación sencillos. Está diseñado para escalar desde un sólo servidor a miles de máquinas, que ofrecen localmente computación y almacenamiento” (Apache, 2019, p. 2) y es empleado por compañías como el propio Yahoo! o Facebook.

Parte del framework MapReduce para procesar los grandes volúmenes de datos y para almacenar los datos emplea un el sistema de ficheros distribuido Hadoop Distributed File System (HDFS). “Es altamente tolerante a fallos y está diseñado para ser desplegado sobre un hardware low-cost. Proporcionando un alto rendimiento en el acceso a aplicaciones de datos y es adaptable a aplicaciones con grandes conjuntos de datos” (Beakta, 2015, p.215) .

Apache Hadoop ha formado un ecosistema compuesto de distintas librerías construidas sobre su sistema de almacenamiento de ficheros HDFS. Entre las destaca Mahout, dedicada a la implementación y ejecución de técnicas de Aprendizaje Automático o *Machine Learning.*

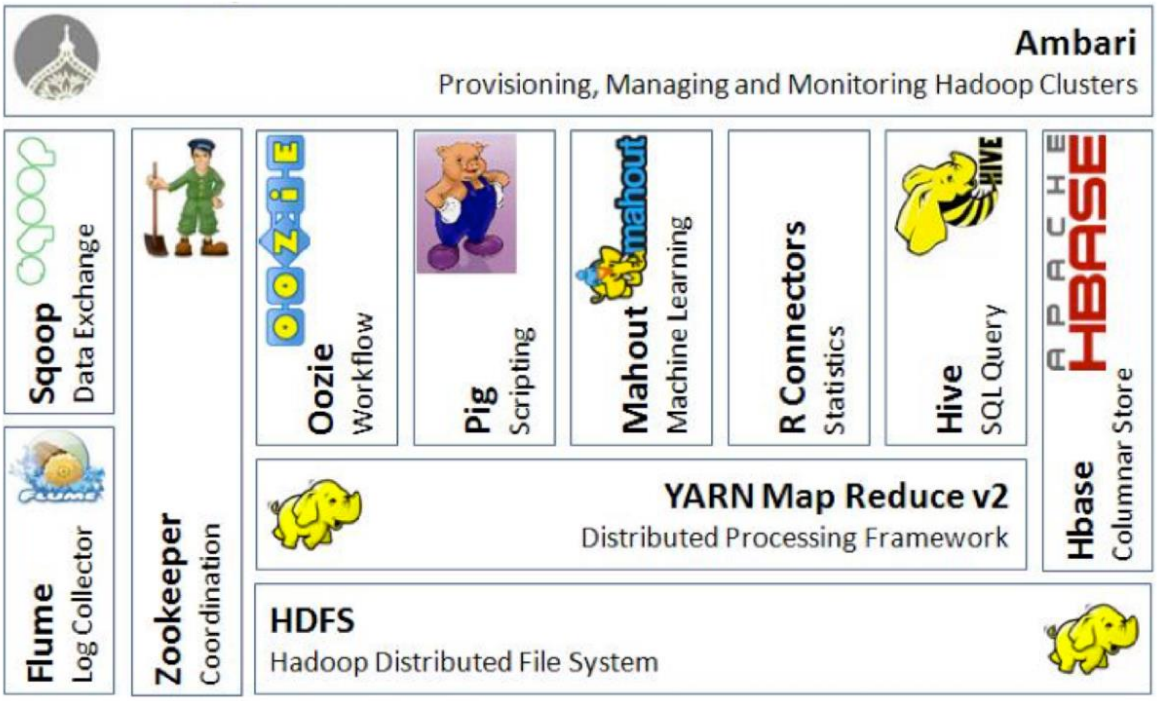

Figura 2.3 Ecosistema Apache Hadoop (Opensource.com, 2014)

De acuerdo con Lin (2012) y, como ya se ha comentado, MapReduce posee grandes ventajas como la capacidad de escalar horizontalmente grandes cantidades de datos empleando miles de

servidores con buena relación calidad-precio, su semántica de programación es fácil de comprender y proporciona una alta tolerancia a fallos.

2.2.2 Apache Spark

Tras el éxito de Hadoop nace Spark para superar los inconvenientes que este presentaba. Fue creado 2009 en la UC Berkeley AMPLab, en 2010 se lanzó como código abierto y en 2013 pasó a formar parte de Apache Software Foundation (History | Apache Spark, s. f.).

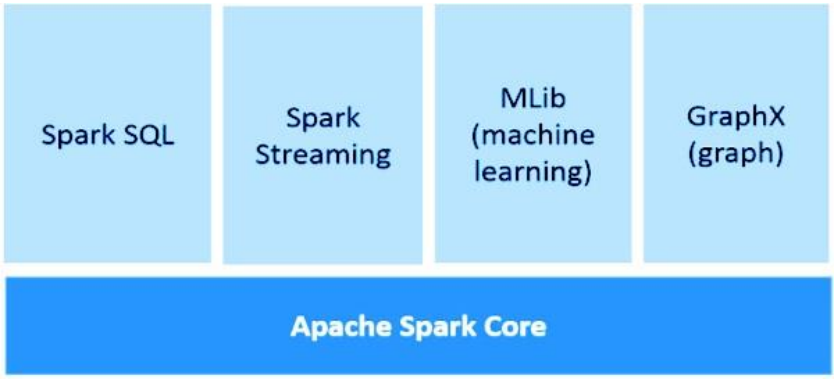

Figura 2.4 Ecosistema Apache Spark (Sitio big data, 2019).

Los frameworks hasta entonces carecían de abstracciones para aprovechar la memoria distribuida y esto los hacía ineficientes a la hora de trabajar con algoritmos iterativos de Machine Learning (Zaharia et al., 2012). Por lo que crearon una nueva abstracción en memoria llamada Resilient Distributed Datasets (RDD) e integrada en Spark.

Los RDDs son tolerantes a fallos y permiten al usuario mantener los resultados en memoria, empleando esta estructura de datos en paralelo para luego emplearlos en procesos futuros. Controlan las divisiones que se hacen de los datos originales para optimizar su almacenamiento y poder manipularlos mediante operaciones más elaboradas (Zaharia et al., 2012).

“Formalmente son una colección dividida de registros, que sólo pueden ser creadas a través de operaciones deterministas sobre los datos en un almacenamiento estable o sobre otros RDDS” (Zaharia et al., 2012, p. 17). Sobre ellos se pueden realizar operaciones de transformación (las explicadas aquí) y operaciones de acción, aquellas que devuelven un valor al proceso aplicado sobre los datos o que exportan los datos a un sistema de almacenamiento (Zaharia et al., 2012).

Spark está programado en Scala y trata los RDDs como objetos dentro de su API. Los transforma mediante métodos que son invocados sobre estos objetos (los RDDs) y la primera vez que se emplean para realizar una acción sobre los datos los computa de forma perezosa (Zaharia et al. 2012).

Desde hace unos años, a partir del lanzamiento de Spark 1.6, se emplea una nueva colección de datos distribuida (*Datasets*) organizados en columnas o atributos con un nombre, denominados *Dataframes*. Son equivalentes a las tablas de las bases de datos relacionales y a los dataframes (existentes en R o Python). Estos *Dataframes* pueden construirse a partir de una amplia variedad de recursos, como se explica en la documentación de Spark (Spark SQL and DataFrames - Spark 2.3.0 Documentation, s. f.).

En su funcionamiento, al igual que Hadoop, trabajan con la esencia de la filosofía MapReduce pero mejoran algunos aspectos de su funcionamiento. De forma que permiten superar las limitaciones de MapReduce sobre algoritmos iterativos de Machine Learning o la necesidad de aplicar un Map-Reduce-Reduce, imposible de hacer con MapReduce. Además, tiene implementaciones en Java, Python, Scala y R.

Todo esto no puede hacernos pensar en que son tecnologías rivales, lo cierto es que Hadoop y Spark se complementan. Spark puede instalarse sobre el Hadoop Distributed File System (HDFS) si es necesario. Aunque algo que ha hecho que Spark sea tan utilizado para el tratamiento de datos en Big Data, es que permite procesar datos en tiempo real.

3 Proceso de KDD en entornos de Big Data

En los apartados anteriores se ha descrito de forma genérica el proceso de KDD de forma genérica. Por lo que ahora se pasa a hablar de él aplicado a entornos Big Data (que es el tipo de datos que motiva este trabajo).

En general, las fases del proceso de KDD y las técnicas aplicadas en cada una de ellas difieren muy poco de lo que se ha expuesto. Aunque, sí es cierto, que con el aumento del volumen de datos, la mayoría de técnicas se adaptarán y también se aplicarán técnicas nuevas.

En este proceso, se deben considerar las cinco V que caracterizan el tipo de datos con el que se trabaja, el Big Data, que son: volumen, variabilidad, valor, velocidad y variedad (actualmente, existen autores que las amplían hasta a siete). A continuación se comentan más detalladamente:

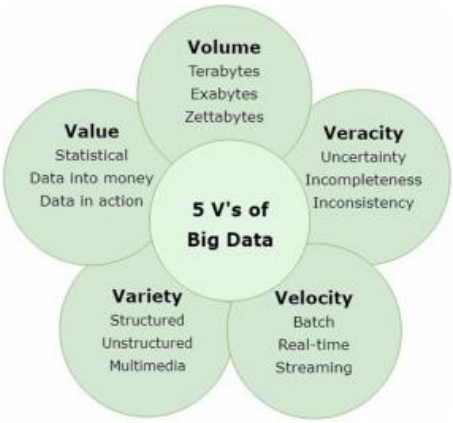

Figura 3.1. Las 5 Vs del Big Data (Hariri et al., 2019).

- Volumen: representa la cantidad de datos generada por segundo (Saguna, 2016).

- Velocidad: se relaciona, como su nombre indica, con la velocidad a la que se presentan los datos al usuario final (Saguna, 2016)..

- Variedad: tiene que ver con la complejidad de los datos, es decir, con la diversidad de tipos de datos con la que se trabaja. Estos datos, a su vez, pueden provenir de fuentes muy diversas como datos generados en tiempo real, bases de datos relacionales, bases de datos no relacionales, redes sociales, gráficos o la Web (Saguna, 2016).

  Asimismo, se puede trabajar con datos almacenados en bases de datos estructuradas (bases de datos tradicionales, se basan en el concepto de entidad-relación) o en bases de datos no estructuradas (contienen redundancias, se rompe el concepto anterior de entidad-relación) (Saguna, 2016).

- Veracidad: los datos son veraces y se puede confiar en ellos (Saguna, 2016).

- Valor: los datos tienen valor estadístico para poder sacar conclusiones de ellos (Saguna, 2016).

3.1 Fases del proceso de KDD en entornos Big Data

En primer lugar, de nuevo hay que destacar la importancia de la fase de preprocesamiento del proceso de KDD, en este caso, en entornos Big Data. Por este motivo, se detallarán y se explicarán detenidamente las distintas más relevantes a aplicar cuando se trabaja con grandes volúmenes de datos.

- Reducción de dimensionalidad, según Luengo et al. (2020), cada una de las cinco Vs que definen al Big Data esconde un gran reto. El volumen de los datos ha sobrepasado la capacidad de almacenamiento y procesamiento de las máquinas y algoritmos. Por este motivo, la reducción de la dimensionalidad se ha vuelto imprescindible para combatir el problema de la maldición de la Gran Dimensionalidad. Consiste en disponer de un gran volumen de datos desde el punto de vista longitudinal o *Big Dimensionality* y se busca eliminar todos aquellos atributos que no son importantes o que resultan redundantes en el conjunto de datos.

  De acuerdo con Luengo et al., (2020), existen muchas formas de describir la realidad y el experto puede considerar que un determinado número de atributos la representa mejor en base a su experiencia o forma de entender los datos de entrada. Por ello, el número de atributos puede aumentar considerablemente y puede llevar al problema, comentado en el apartado anterior, de pérdida de poder de discriminación entre atributos y de pérdida de poder de predicción de las variables explicativas.

  Se hace muy necesario emplear técnicas de selección y de extracción atributos que eliminen las instancias que introducen ruido en los datos o que tienen poco poder de discriminación. Ayuda a que los algoritmos de Minería de Datos se desarrollen correctamente, de forma más rápida y con mayor precisión en sus resultados. También reducen el espacio de almacenamiento necesario para almacenar el conjunto de datos y la memoria requerida para realizar el proceso (como se ha comentado en apartados anteriores).

  Los algoritmos distribuidos de mRMR, *conditional mutual information maximization* (CMIM) y *joint mutual information* (JMI) proporcionan no sólo una buena formulación teórica, si no que también un buen ajuste en plataformas de Big Data, de acuerdo con Luengo et al., (2020).

- Reducción de instancias o datos, siguiendo a Luengo et al. (2020), se compone de un conjunto de técnicas que reducen el tiempo de ejecución, el espacio de almacenamiento y aceleran y mejoran la fase posterior de Minería de Datos. Retienen la información más importante de los datos a través de la eliminación de instancias irrelevantes, que introducen ruido o son redundantes.

  Emplean algoritmos para la limpieza y simplificación de los datos, ya que las técnicas de reducción de instancias no son aplicables en entornos Big Data (Luengo et al., 2020). Su objetivo (como se ha comentado más arriba) es obtener un subconjunto de datos libres de instancias problemáticas, que se empleará como conjunto de entrenamiento en la fase de Minería de Datos. Para ello puede valerse de técnicas de generación de instancias de forma artificial para obtener una mejor representación de los datos de entrenamiento, de acuerdo con Luengo et al. (2020).

  MapReduce posee un *framework* distribuido llamado MRPR para desarrollar las técnicas de reducción de prototipos (*prototype reduction* o PR), según Triguero et al. (2015) (Citado en Luengo et al., 2020). Esto lo hace mediante un proceso de estratificación de instancias de forma paralela. La función *map* divide los datos y contiene la aplicación de esta técnica, creada para superar las debilidades del algoritmo k-vecinos más cercanos (*k- nearest neighbours* o KNN) (como la mayoría de estas técnicas de reducción de instancias). La función reduce, por su parte, filtra o fusiona los prototipos para dejar libre de valores negativos el conjunto final empleado para entrenar el modelo, de acuerdo a las ideas expuestas por Luengo et al. (2020).

- Datos imperfectos, como se ha comentado, se componen de técnicas de filtrado del ruido y de técnicas de imputación de valores perdidos. Ambas son muy importantes debido a que el ruido afecta a las etiquetas de las clases, desdibuja los límites del problema y reduce el rendimiento de los clasificadores. Los valores perdidos también generan problemas que afectan al proceso de aprendizaje del algoritmo, ya que estos esperan recibir como entrada datos completos, de acuerdo con lo expuesto en relación a este tipo de datos por Luengo et al. (2020).

  El volumen de datos que se maneja en entornos Big Data hace imposible emplear las técnicas tradicionales de limpieza de datos imperfectos. Pueden existir instancias

redundantes de forma no balanceada que pueden generar solapamiento entre clases o atributos. Por ello se necesita realizar un estudio específico de cada región que requiere tratamiento.

En Big Data, García-Gil et al. (2019) (Citado en Luengo et al., 2020) proponen un *framework* bajo Apache Spark para aplicar técnicas de filtrado de ruido en problemas de clasificación. Se compone de dos algoritmos: *Homogeneous Ensemble for Big Data (HME-BD)* (divide el conjunto de entrenamiento mediante un clasificador Random Forest) y *Heterogeneous Ensemble for Big Data (HTE-BD)* (que identifica instancias ruido mediante clasificadores Random Forest, de regresión logística y k-vecinos más cercanos (KNN)). A estos dos algoritmos, los autores suman el algoritmo *Edited Nearest Neighbor for Big Data (ENN-BD)* (basado en la similitud entre instancias del conjunto de entrenamiento, elimina aquellas que tienen un gran número de vecinos pertenecientes a distintas clases).

Según estos autores (Luengo et al., 2020), también existen técnicas de limpieza de los datos que se basan en el algoritmo k-vecinos más cercanos (KNN). De esta forma, detecta las instancias que pertenecen a un gran número de clases y las elimina del conjunto de datos original, ya que se consideran ruido.

Como se observa, en la eliminación de ruido se diferencian dos ramas: una basada en conjuntos de clasificadores y otra que parte del algoritmo k-vecinos más cercanos (KNN), de acuerdo con Luengo et al. (2020). Esta última rama, en concreto, ha sido muy prolífica en Big Data.

- Discretización, de acuerdo con Luengo et al. (2020), los datos se presentan en distintos formatos. Esto determina (junto al dominio de los datos) el tipo de algoritmo de Minería de Datos aplicable. Los valores categóricos o nominales no presentan un orden, a diferencia de los valores numéricos (ya sean discretos o continuos).

  Se trata de una técnica muy empleada en los últimos años, debido a que es una de las técnicas de preprocesamiento mayor efectividad (García et al., 2013; García et al., 2015; Liu & Motoda 2002) (Citados en Luengo et al. 2020). Consiste en transformar un gran conjunto de datos que contiene valores numéricos en un conjunto de datos menor, que contenga valores categóricos (puede decirse que transforma atributos cualitativos en

atributos cuantitativos). Sin incurrir en el solapamiento del dominio de los intervalos en los que divide las instancias.

En Big Data existen (aunque no son muchos) distintos métodos de discretización insertados en entornos de programación en paralelo. Esto se debe a la complejidad en la que incurren estos algoritmos, que va de log-linear en aumento, de acuerdo con Luengo et al. (2020). Autores como Cerquides et al. (1997), Zhao et al. (2011) y Cano et al. (2014) son los que han propuesto distintos enfoques de estos métodos.

Los primeros autores (Cerquides et al., 1997) (Citado en Luengo et al., 2020) propusieron un método basado en la distancia de Mantaras para evaluar si las divisiones de los datos son las correctas. Desarrolla las fases de clasificación u ordenación, evaluación de los puntos y división (entre otras) de forma paralela. Años más tarde, Parthasarathy y Ramakrishnan (2002) (Citado en Luengo et al., 2020) desarrollaron una extensión para datos generados en tiempo real.

Zhao et al. (2011) (Citado en Luengo et al., 2020), posteriormente, desarrollaron un método de discretización en paralelo basado en la idea del z-score. Se trata de un algoritmo que emplea un rango dinámico para reflejar la importancia de las distribuciones de probabilidad de los atributos a discretizar.

Finalmente, Cano et al. (2014) (Citado en Luengo et al., 2020), presentaron un algoritmo de discretización en paralelo que emplea la GPU (*graphics processing unit* o unidad de procesamiento gráfico). Para ello se inspira en el discretizador CAIM y es capaz de paralelizar un gran volumen de operaciones.

Una vez se ha finalizado con éxito esta fase del proceso de KDD, se habrán obtenido datos de calidad o *Smart Data* que se relacionan con dos conceptos que caracterizan al Big Data, definidos anteriormente, que son la veracidad y el valor. Ambos permiten la aplicación de algoritmos y facilitan la obtención de conocimiento y resultados de calidad que puedan utilizarse para tomar decisiones de manera inteligente (García et al. 2016). El fin último por el que se inicia este proceso.

Además de estas dos características, siguiendo a García et al. (2016), la exactitud, el ser procesables y ágiles definen a estos datos objetivo. Los datos deben ser lo se de dice que son,

deben ser escalables para su posterior procesamiento y deben también estar disponibles y listos para adaptarse a los cambios que puedan surgir.

Una vez se ha obtenido un conjunto de datos libre instancias y atributos que puedan generar problemas al aplicar algoritmos de Minería de Datos. En esta siguiente fase de Minería de Datos, al trabajar en entornos Big Data, las técnicas que se apliquen deben enfrentarse a grandes volúmenes de datos heterogéneos, interconectados y que se generan en tiempo real. Además, deben enfrentarse a retos a la hora de manejarlos con estas técnicas. Por lo que es imposible aplicar los algoritmos y las técnicas tradicionales de Minería de Datos.

Los retos que supone el Big Data a la hora de procesar los datos llevan a que se necesiten nuevas tecnologías, algoritmos avanzados y una gran variedad de nuevos enfoques a gran escala. Todo esto deberá tenerse en cuenta a la hora de implementar las técnicas de Minería de Datos De este modo, los retos de procesamiento que supone el Big Data son:

- Heterogeneidad, la variedad de los datos y la infinidad de fuentes de los que provienen (Ahsaan & Mourya, 2019).

- Puntualidad, se necesitan manejar y analizar los datos a gran velocidad. La velocidad a la que se generan los datos y su rápida pérdida de valor hacen que se deban procesar en tiempo real. Muchas veces, esto lleva a que se implanten procedimientos incompletos y se obtengan resultados menos refinados (Ahsaan & Mourya, 2019).

- Complejidad, se relaciona principalmente con los datos no estructurados y con la rigidez de estos, debido a que se relacionan con contenidos multimedia como fotos, vídeos o publicaciones en redes sociales (Ahsaan & Mourya, 2019).

- Escalabilidad, los datos crecen muy rápidamente de forma diaria. Este gran volumen de datos generados a gran escala sólo es procesable si se mejora la velocidad que posee el procesador (Ahsaan & Mourya, 2019). Por este motivo, surgen tecnologías como MapReduce (comentada en este trabajo), que trata de reducir el problema de que la velocidad a la que generan los datos sea mayor a la velocidad a la que pueden procesar los ordenadores. Esto sólo lo consigue añadiendo nuevos ordenadores o servidores que ayuden al procesado de los datos, es decir, se vuelve en una tecnología escalable.

- Precisión, se relaciona con la precisión de los datos y con el número limitado de fuentes de los que provienen (Ahsaan & Mourya, 2019). Estas deben ser auténticas para que los resultados del análisis sean del mismo modo precisos y auténticos.

De acuerdo con Che et al. (2013), una posibilidad sería mejorarlos empleando para ello arquitecturas de programación en paralelo de forma masiva. A esto se une el nacimiento, en los últimos años, de plataformas de programación distribuida como Apache Hadoop, Apache Spark, o Giraph. Por lo que la Minería de Datos aplicada a entornos Big Data es una disciplina con un gran potencial y de la que aún queda mucho por desarrollar. A continuación detallaremos algunas técnicas que se pueden desarrollar cuando se trabaja en estos entornos.

- Asociación, son conjunto de técnicas que identifican patrones en los datos a través de un estudio de la correlación entre diversos atributos.

- Clustering, identifica grupos de instancias que poseen las mismas características. Emplean algoritmos de similitud o disimilitud entre grupos de instancias. Estos se basan en calcular la distancia máxima o mínima entre la instancia y un punto dado.

- Árboles de decisión, se emplean tanto en problemas de regresión como de clasificación. Se representan gráficamente mediante nodos, cada uno recoge una pregunta o respuesta que caracteriza a los datos.

- Clasificación, buscan predecir la clase a la que pertenece a una determinada instancia. Toman como referencia las características de la instancia y la comparan para ver sus similitudes con las características de las clases (que vienen dadas).

En la aplicación de cualquiera de estos algoritmos se diferencian dos fases (como se ha comentado en apartados anteriores). La primera fase de entrenamiento del modelo consiste en dividir el conjunto de datos original en dos subconjuntos, uno de entrenamiento (que, por ejemplo, supondrá en torno al 80% de los datos originales) y uno de prueba o test (que, siguiendo el ejemplo, supondrá el 20% de los datos restante). Este conjunto de entrenamiento servirá para que el algoritmo aprenda los datos y pueda generar un primer resultado en base a datos conocidos.

Si los resultados arrojados en esta fase no son adecuados, se debe volver a la fase de preprocesamiento de los datos para limpiar y preparar los datos. Una vez se hayan limpiado, se

vuelve a estimar el modelo para el conjunto de entrenamiento y se comprueban los resultados. Este proceso se repetirá tantas veces como sea necesario hasta obtener un modelo que ajuste bien a los datos y genere resultados apropiados en base a los objetivos que se quieren alcanzar.

El objetivo de la fase de Minería de Datos es obtener conocimiento nuevo a partir de datos desconocidos, por lo que es primordial obtener buenos resultados en la fase de prueba. Esta fase sirve para validar el modelo y comprobar su capacidad de generalización a datos nuevos y desconocidos. Si se obtiene un buen ajuste en esta fase, el usuario podrá emplear este modelo para extraer conocimiento a partir de nuevos datos desconocidos.

Finalmente, una vez se haya logrado el éxito en la fase anterior, se pasa a la fase de interpretación de los datos. En esta fase se deberán tener en cuenta las características de los datos, del entorno en el que se trabaja, el contexto y los objetivos del proceso de KDD. Para facilitar la interpretación, el usuario puede valerse de técnicas de visualización de los datos (gráficos de series temporales, gráficos de tarta, gráficos de barra, etc.).

3.2 Análisis Exploratorio de Datos en entornos Big Data

En un Análisis Exploratorio de Datos el analista juega un papel fundamental. Se encarga de explorar los datos desde tantos puntos de vista y de tantas formas como le sea posible. De este modo, el analista no parará hasta encontrar información que sea verdaderamente relevante en los datos. Por lo que deberá mantenerse abierto a cualquier resultado que puedan arrojar las distintas fases de este análisis.

El problema es que se hace imposible aplicar las técnicas tradicionales de Análisis Exploratorio de Datos cuando el tamaño de las bases de datos se vuelve demasiado grande. Por este motivo, surge una nueva disciplina denominada *Big Data Analytics* o *Big Analytics*.

"*Big Data Analytics* describe el proceso de analizar bases de datos masivas para descubrir patrones, correlaciones desconocidas, tendencias del mercado, preferencias del usuario y otra información de valor que no podría ser analizada con las herramientas tradicionales" (Golcha, 2015. Citado en Hariri et al., 2019, p.7).

Al igual que cuando se ha descrito la fase de Minería de Datos del proceso de KDD en entornos Big Data surgían retos al trabajar con este tipo de datos. Cuando se realiza un análisis exploratorio empleando *Big Data Analytics*, los datos también imponen ciertos retos a estas técnicas.

- *Retos en el almacenamiento de los datos,* almacenar petabytes o exabytes de datos generados día a día requiere un espacio de almacenamiento con un tamaño proporcional al tamaño de estos datos, (Vaidya & Kshirsagar, 2020). Los clústers de servidores generalmente solventan este problema, pero surgen otros a la hora de extraerlos, agruparlos, estructurarlos y aplicarles técnicas de visualización (Raghav et al, 2015; citado en Vaidya & Kshirsagar, 2020).

- *Retos en el manejo de los datos.* De acuerdo con Vaidya & Kshirsagar, (2020), al trabajar con Big Data se pueden encontrar datos en muy diversos formatos. Centralizarlos en una base de datos para luego volverlos a descentralizar es una tarea que se vuelve muy complicada. Además, durante esta tarea pueden fallar los sistemas que manejan y transportan a las bases de datos tradicionales desde sus fuentes originarias.

- *Retos en el procesamiento de los datos.* Siguiendo a Vaidya & Kshirsagar, (2020), antes de ser procesados los datos, se necesita aplicar herramientas de ETL (en inglés, *extract, transform and load*) como Talend. Una vez se han limpiado y se les ha dado significado a los datos, será más fácil su posterior procesamiento.

- *Retos en la optimización de consultas.* De acuerdo con Vaidya & Kshirsagar, (2020) los datos almacenados deben generar resultados importantes para que las predicciones y procesamientos a los que se sometan sean más sencillos. Se han creado entornos que permiten extraer resultados con valor y significado para el usuario de la consulta y de forma rápida (Wani & Jabin, 2017; Doulkeridis & Nørvåg, 2013) (Citados en Vaidya & Kshirsagar, 2020). Existiendo también algoritmos más avanzados para manejar enormes conjuntos de datos.

- *Retos en la seguridad de los datos.* Siguiendo a Vaidya & Kshirsagar (2020) los datos de los que dispone el analista muchas veces contienen registros de transacciones y una gran cantidad de información personal de los individuos que los generaron. Por este motivo,

debe velar por la seguridad de los mismos durante el almacenamiento y manejo de los mismos.

- *Retos en el diseño de los entornos de programación*. Según Vaidya & Kshirsagar (2020) existe una gran cantidad de entornos en el mercado. El problema es que no todos son capaces de resolver todos los problemas a los que se enfrenta el usuario de las técnicas de *Big Data Analytics*. Por este motivo, deben desarrollarse algoritmos que aporten precisión a la hora de implementarlas utilizando estos algoritmos.

- *Falta de expertos en datos* debida a que los datos crecen tan rápido que no da tiempo a que los usuarios se conviertan en expertos de su manejo, sobre todo en el caso de conjuntos de datos muy pesados (Vaidya & Kshirsagar, 2020).

- *Retos en analíticas,* los distintos tipos de técnicas de *Big Data Analytics* (que se describen en el siguiente apartado) se aplican según las características de los datos. Por lo que existen distintos tipos de estrategias de análisis y los analistas deberán ser quienes decidan cuál es más adecuada en base a ello (Vaidya & Kshirsagar, 2020).

- *Retos en la calidad de los datos,* que, como se ha comentado, se presentan en distintos formatos y provienen de distintas fuentes. De acuerdo con Vaidya & Kshirsagar (2020), para poder mejorar su calidad se hace necesario estructurarlos muy bien. Por lo que mantener una buena calidad, se vuelve un reto para los desarrolladores.

- *Retos en la conversión de Big Data en insights con valor,* es un proceso crucial y muy pocas tecnologías son capaces de generar información de valor. Por otro lado, ahorrará tiempo y dinero durante el desarrollo de estas técnicas si el usuario dispone de datos importantes de una buena fuente (Vaidya & Kshirsagar, 2020)

- *Otros retos técnicos relacionados con el tiempo de ejecución*. De acuerdo con Vaidya & Kshirsagar (2020), surgen cuando se trabaja con datos generados en tiempo real. Las librerías y las técnicas de preprocesamiento de los datos deberán estar listos para trabajar con estos datos. De otro modo, se generarán problemas técnicos durante el proceso.

3.2.1 Técnicas de Análisis Exploratorio de Datos en entornos Big Data.

Pese a los restos que supone trabajar en entornos Big Data, existe una clasificación más o menos bien diferenciadas de las técnicas de *Big Data Analytics* existentes. De modo que siguiendo la definición dada anteriormente, Riahi & Riahi. (2018) proponen cuatro tipos de técnicas de *Big Data Analytics*, que son las que se definen a continuación:

- *Descriptive Analytics o Analíticas Descriptivas.* De acuerdo con Riahi & Riahi. (2018), responden a la pregunta: ¿qué está pasando? Es el primer paso, anterior al procesamiento de los datos, que crea un conjunto de datos históricos. Estas técnicas generan probabilidades futuras y tendencias que sirven para dar una idea sobre qué pasará en el futuro.

- *Diagnostic Analytics o Analíticas de Diagnóstico.* Siguiendo a Riahi & Riahi. (2018), responden a la pregunta: ¿por qué ha pasado? Se emplean para determinar y entender el motivo que ha generado un evento o comportamiento.

- *Predictive Analytics o Analíticas Predictivas.* Según Riahi & Riahi. (2018), responden a la pregunta: ¿qué puede pasar?, empleando para ello datos del pasado. Se basan en la predicción y "utilizan técnicas de Minería de Datos e Inteligencia Artificial para analizar los datos actuales y establecer escenarios futuros" (Riahi & Riahi., 2018, p. 3).

- *Prescriptive Analytics o Analíticas Prescriptivas.* De acuerdo con Riahi & Riahi. (2018), responden a la pregunta: ¿qué debe hacerse? Ayudan a predecir el futuro y buscar la acción correcta que se debe implementar.

Por otro lado, de acuerdo con Thakur et al. (2020), también existen técnicas de *Big Data Analytics* basadas en algoritmos, que ayudan a extraer la información necesaria de los datos y que permiten conocerlos mejor. Entre ellas, se pueden nombrar las siguientes:

- *Aprendizaje mediante reglas de asociación.* De acuerdo con Thakur et al. (2020), ayuda a decidir sobre el tipo de relaciones presentes en los datos en base a su valor. Para ello asocian una probabilidad de ocurrencia de cada relación posible que pueda existir.

- *Minería de Datos,* es una disciplina con diversas aplicaciones en campos muy diferentes. En *Big Data Analytics* ayuda a inferir modelos que diseccionan la información presente en los datos en distintos grupos mediante al menos un lenguaje de programación (Thakur, et al., 2020).

- *Análisis basado en clústers,* ayuda a identificar instancias con características similares para, por ejemplo, identificar patrones en los datos (Thakur, et al., 2020).

3.2.2 Recursos para el Análisis Exploratorio de Datos en entornos Big Data.

En entornos Big Data existen toda una serie de recursos o herramientas informáticas que permiten, con un mejor o peor desempeño, completar con éxito el Análisis Exploratorio de Datos. Estas herramientas no sólo son del ámbito exclusivo de la Ciencia de los Datos o la Computación, si no que, al estar muy relacionado este análisis con la Estadística, también aparecen herramientas de esta disciplina.

R es la herramienta o recurso por excelencia utilizada en Ciencia de los Datos para el tratamiento de las bases de datos desde el punto de vista científico. Es un entorno de programación de código abierto para Estadística y está disponible en un gran número de sistemas operativos.

Dispone de numerosas librerías o paquetes que permiten automatizar o acelerar el proceso de Análisis Exploratorio de Datos. Entre estas librerías destacan arsenal, autoEDA o DataExplorer (Staniak & Biecek, 2019).

Otra herramienta muy utilizada en el campo de la Estadística y Econometría es STATA, aunque también es utiliza en Ciencia de los Datos. Este software estadístico trabaja bajo licencia y permite manejar y visualizar datos, implementar técnicas estadísticas y automatizar informes (StataCorp LLC, s.f.). Está programada en C y la última versión estable lanzada por la empresa SataCorp LLC es Stata 16.1. Stata dispone de varias licencias y todas ellas pueden ser ejecutadas en cualquier máquina.

Stata/IC y Stata/SE, están desarrolladas para ser utilizadas si se dispone de conjuntos de datos de mediano tamaño (2.048 variables) y gran tamaño (32.767 variables) con 2,14 billones (americanos) de observaciones, respectivamente.

Stata/MP, por su parte, se desarrolla para manejar conjuntos de datos de mayor tamaño en un servidor de gran capacidad y es la licencia que presta mayor velocidad. El número máximo de observaciones que permite trabajar es de 20 billones americanos almacenadas hasta en 120.000 variables. Aunque presenta el problema de que no es escalable, es decir, no puede crecer añadiendo más servidores a esta licencia. Por lo que no podemos decir que en realidad sea una herramienta que siempre pueda utilizarse para realizar un Análisis Exploratorio de Datos en entornos Big Data.

El entorno de programación empleado en este trabajo, Apache Spark, también dispone de elementos en su ecosistema que permiten crear recursos para aplicar herramientas de Análisis Exploratorio de Datos. En concreto, este elemento es GraphX y se trata de una herramienta diseñada puramente para trabajar en entornos Big Data.

De acuerdo con Luengo et al. (2020), GraphX permite procesamiento gráfico de los datos mediante una interfaz única, utilizando la tecnología de Apache Spark. Es capaz de unificar todo el proceso de extracción, transformación y carga de los datos (en inglés, *extract, transform and load* o, más conocido como, ETL), el proceso de aprendizaje y el análisis exploratorio de los datos en la API de Spark. Además, permite al usuario diseñar gráficos iterativos mediante la API de Pregel (Google).

Finalmente, otro de los softwares más utilizados en Ciencia de los Datos MATLAB, que dispone de *toolboxes* como MEDA (*Multivariate Exploratory Data Analysis*) para el desarrollo del Análisis Exploratorio de Datos o EDA (sus siglas en inglés). MATLAB (abreviatura de *Matrix Laboratory*) es un *software* matemático desarrollado por MathWorks, está programado en C, Java y lenguaje MATLAB y está disponible en una amplia variedad de sistemas operativos.

El *toolbox* MEDA está desarrollado por José Camacho, Rafael Rodríguez, Alejandro Pérez y Elena Jiménez-Mañas, profesores de la Universidad de Granada. De acuerdo con Camacho et al. (2015), está compuesto por un conjunto de herramienta para el análisis multivariante que sirven para realizar análisis exploratorio de los datos. Emplea técnicas tradicionales de exploración de los datos mediante gráficas como el Análisis de Componentes Principales o Mínimos Cuadrados Parciales. Más conocidos por sus siglas en inglés PCA (*Principal Component Analysis*) y *Partial Least Squares* (PLS).

Es una herramienta que puede ser utilizada tanto para el análisis de bases de datos de tamaño regular como para el análisis de bases de datos de mayor tamaño, con millones de elementos (Camacho et al., 2015). Para ello emplea una extensión que trabaja en entornos Big Data.

3.3 Bibliotecas para Machine Learning en entornos de Big Data

El desarrollo de las tecnologías necesarias para Big Data y los modelos de programación distribuida han permitido el almacenamiento y procesamiento de Big Data. Por ello, en relación al proceso de KDD, se ha hecho necesario desarrollar proyectos que permitan implementar los algoritmos de Machine Learning que permiten la extracción de conocimiento a partir de estos datos.

Las dos tecnologías descritas, Apache Hadoop y Apache Spark, poseen librerías que dan solución a muchos de los procesos que ponen las fases del proceso de KDD (como limpieza de datos, entrenamiento de algoritmos, validación de algoritmos, etc).

Estas librerías son Apache Mahout, integrada en el ecosistema Hadoop, y Spark MLlib, integrada en el ecosistema Apache Spark y ambas, al igual que los entornos a los que pertenecen, son librerías de código abierto. A continuación se describirán más detalladamente.

3.3.1 Mahout

Es la biblioteca escalable englobada en el ecosistema Hadoop. Se dedica a la implementación de algoritmos de Machine Learning y Minería de Datos de todo tipo: clasificación, *clustering*, regresión y recomendación. Almacena los datos en HDFS o en memoria, su uso es de propósito general y las implementaciones de MapReduce escalan de forma lineal con los datos (Withanawasam, 2015).

El problema es que no todos los algoritmos de Minería de Datos pueden implementarse de manera eficiente con MapReduce. Siguiendo a Lin (2012), los algoritmos de gráficos iterativos (como el algoritmo de Google, PageRank), los de *expectation maximization* (como el algoritmo *k-means*) o los de gradiente descendente (como el clasificador de regresión logística) no generan un buen ajuste si se emplea MapReduce.

Para superar este problema, se emplean otro tipos de entornos de programación como Pregel, Spark o Giraph. Sin olvidar que ningún modelo de programación o entorno será el mejor para resolver todos los problemas, siempre se deberá buscar el equilibrio entre la simplicidad, tolerancia a fallos, rendimiento, facilidad de uso, etc. (Lin, 2012).

De acuerdo con Withanawasam (2015), Apache Mahout es apropiado para aplicar algoritmos con un propósito industrial y es crítico el desempeño del algoritmo, si se busca una solución de código abierto o si los datos crecen muy rápido y alcanzan grandes volúmenes. También lo es si se prefiere un procesamiento de los datos por lotes o batches o si se busca una librería consolidada en el mercado.

3.3.2 MLlib

Es una librería que resulta muy interesante en este trabajo, pues permite desarrollar el proceso de la extracción de conocimiento a partir de grandes volúmenes datos con Spark y trabaja sobre RDDs.

Esta librería nace en 2012 y en 2013 se lanza como opción de código abierto integrada dentro de la licencia de Apache 2.0. Está escrita en Scala e incluye APIs para Java, Scala, Python y R. Al igual que Spark, es de código abierto y posee una gran comunidad que desarrolla los métodos y herramientas incluidos en ella (Meng et al., 2016).

Recoge un compendio de algoritmos tanto de aprendizaje supervisado como no supervisado y de técnicas de limpieza de datos como la reducción de la dimensionalidad. Todos ellos muy necesarios en las etapas de preprocesamiento y de Minería de Datos del proceso de KDD. Asimismo, permite crear pipelines o tuberías, algoritmos de persistencia y modelos a partir de ellos (Veith & Assunçao, 2019).

Las principales características que definen esta librerías son:

- *Métodos y utilidades compatibles*: esta librería proporciona implementaciones rápidas y distribuidas de los algoritmos de Machine Learning más conocidos como modelos de álgebra lineal, tanto para problemas de clasificación como de regresión. Dos ejemplos de estos modelos son el clasificador Naïve Bayes o los árboles de decisión (Meng et al., 2016).

Además, proporciona utilidades básicas para optimización convexa, álgebra lineal distribuida, análisis estadístico o extracción de características. Soporta también distintos formatos de entrada y salida como LIBSVM, integración de datos mediante Spark SQL o el formato interno de MLlib para exportar estos modelos (Meng et al., 2016).

- *Optimización de algoritmos*: ayudan en la eficiencia del aprendizaje distribuido y las predicciones realizadas. Existe un gran número de optimizaciones dentro de esta librería, entre las que podemos destacar los árboles de decisión o los modelos lineales generalizados (Meng et al., 2016).

- *Pipeline API:* incluyen una secuencia de preprocesamiento de los datos, extracción de características, ajuste de los modelos y validación de los mismos. Se engloba dentro del paquete denominado *spark.ml*, que simplifica el desarrollo y puesta a punto de las pipelines (Meng et al., 2016).

  Se trata (*spark.ml*) de una API de alto nivel que ayuda a crear pipelines complejas en varias etapas para conectar componentes, por ejemplo, de las fases de preprocesamiento, aprendizaje o evaluación. Además, permite realizar, entre otras, la selección de modelos, distintas estrategias de validación de estos modelos o el ajuste de hiperparámetros de los mismos (Meng et al., 2016).

- *Spark Integration*: el núcleo de Spark le provee una máquina de ejecución general, en su nivel más bajo. A un nivel superior, MLlib mejora el rendimiento de otras librerías de Spark como Spark SQL, GraphX o Spark Streaming (Meng et al., 2016).

- *Documentación, Comunidad y Dependencias*: la guía de usuario de Spark MLlib proporciona una extensa documentación que describe todos los métodos y utilidades disponibles en la librería, ilustrando estas explicaciones mediante ejemplos para los distintos lenguajes que soporta. Además, esta guía también incluye las dependencias relativas al código de programación (Meng et al., 2016).

No obstante, desde que se lanzó la segunda versión de la API de Spark, se ha migrado a la librería Spark ML, que trabaja sobre *Dataframes*. A partir de ellos, crea, por un lado, *transformers* o "abstracciones que incluyen transformadores de atributos y modelos entrenados" (Spark ML

Programming Guide - Spark 1.2.2 Documentation, s. f.). Por otro, crea *estimators* o estimadores, "que abstraen el concepto de un algoritmo de aprendizaje o cualquier algoritmo que se ajuste o entrene en datos" (Spark ML Programming Guide - Spark 1.2.2 Documentation, s. f.). Esta biblioteca será la que se utilice en el caso de uso.

4 Un caso de uso

Para ejemplificar el proceso de KDD y el Análisis Exploratorio de Datos o EDA, se va a utilizar un conjunto de datos sencillo y se procederá a aplicar distintas técnicas sobre él como si se tratara de un conjunto de mayor tamaño o procedente de entornos Big Data.

Para realizar este análisis se utilizarán distintas librerías que permitan implementar las técnicas necesarias para analizar y preprocesar el conjunto de datos, y posteriormente, implementar varios algoritmos de clasificación sobre él, verificar su bondad de ajuste y comentar los resultados que se obtengan. Finalmente, se hará una interpretación de estos resultados y se hará una comparativa de la bondad de los distintos modelos de clasificación.

4.1 Descripción de los datos

Para elaborar el estudio se ha utilizado un conjunto de datos muy conocido que contiene datos del Instituto Nacional de Diabetes y Enfermedades Digestivas y Renales de la India, empleado por los los autores Smith et al. en su artículo de 1988 y extraído de la web Kaggle (2016). Con él, se tratará de predecir mediante un diagnóstico si un determinado paciente padece diabetes o no, en base a ciertas mediciones de diagnóstico de los individuos incluidos en este conjunto de datos.

Está integrado por 768 mujeres, de acuerdo con Tarrés et al. (2016), residentes en Phoenix, Arizona (EE.UU.), que pertenecen a la etnia Pima y que, en el momento de la recogida de datos, tenían una edad de al menos 21 años.

El conjunto de datos está compuesto de ocho variables explicativas de tipo numérico y una variable de salida binaria que indica la clase. Todas ellas se detallan a continuación:

- *Pregnancies:* número de embarazos.
- *Glucose:* concentración de glucosa en plasma después de 2 horas de realización del Test de Tolerancia a la Glucosa (mg / dl).

- *BloodPressure*: tensión arterial diástolica (mm/Hg)
- *SkinThickness*: grosor de la piel del tríceps (mm)
- *Insuline*: concentración de insulina sérica a las 2 horas de una prueba de tolerancia a la glucosa (μ U/ml) (Tarrés et al., 2016).
- *BMI: Body Mass Index* o Índice de Masa Corporal (IMC) dada por la fórmula: peso en kg/(altura en m)^2
- *DiabetesPedigreeFunction:* antecedentes familiares o función de pedigrí de diabetes (FPD) (número de familiares / casos con diabetes)
- *Age:* edad en años.
- *Outcome*: variable de salida o clasificatoria. Se trata de una variable que toma los valores 0 o 1. El 0 representa a aquellos pacientes que no padecen diabetes y el 1, a aquellos que sí la padecen, basado en el criterio de la OMS.

4.2 Análisis Exploratorio de Datos sobre el caso de uso.

Al trabajar con este conjunto de datos como si se tratase de un entorno Big Data, utilizaremos la API de Apache Spark para Python. De este modo, se emplearán distintas librerías *pyspark* y se combinarán con la librería *pandas*. Esta última, aunque no se puede emplear en entornos Big Data, se empleará en algunos casos para aplicar técnicas específicas de forma más sencilla.

En primer lugar, se debe crear la sesión en el entorno de Spark para poder empezar a trabajar. Para cargar el conjunto de datos basta con utilizar el método read.csv() de la librería *SparkContext()* de *pyspark* como se muestra a continuación.

```
#Iniciamos el entorno Spark para poder ser utilizado en Jupyter notebook
#Cargamos las librerias que vamos a necesitar
import findspark
findspark.init()
findspark.find()
import pyspark
findspark.find()
```

Cuando se carga el conjunto de datos en el entorno de Spark, se le indica que tome la primera fila del archivo como cabecera y que infiera el esquema de los datos automáticamente.

```
# Cargamos los datos en Spark. Para ello, creamos una sesión e importamos el dataset.
try:
    sc.stop() #paramos la instancia Spark si ya estaba arrancada
except Exception:
    pass

sc= SparkContext()
spark = SQLContext(sc)
df = spark.read.csv('diabetes.csv', header=True, inferSchema=True)
```

Así, el esquema que devuelve, mediante la sentencia *df.printSchema(),* es el siguiente:

```
ESQUEMA DEL CONJUNTO DE DATOS:
root
 |-- Pregnancies: integer (nullable = true)
 |-- Glucose: integer (nullable = true)
 |-- BloodPressure: integer (nullable = true)
 |-- SkinThickness: integer (nullable = true)
 |-- Insulin: integer (nullable = true)
 |-- BMI: double (nullable = true)
 |-- DiabetesPedigreeFunction: double (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Outcome: integer (nullable = true)
```

Figura 4.1. Esquema del conjunto de datos.

La mayoría de variables del conjunto de datos contienen valores numéricos de tipo entero, es decir, son valores discretos. Excepto las variables *"BMI*" y "*DiabetesPedigreeFunction*", que son de tipo doble o continuo.

La sentencia *df.describe()* permite mostrar de forma resumida los principales estadísticos que describen los datos almacenados en las distintas variables del conjunto de datos. Se empleará la función toPandas() para mejorar la visualización de la tabla.

summary	count	mean	stddev	min	max
Pregnancies	768	3.8450520833333335	3.36957806269887	0	17
Glucose	768	120.89453125	31.97261819513622	0	199
BloodPressure	768	69.10546875	19.355807170644777	0	122
SkinThickness	768	20.536458333333332	15.952217567727642	0	99
Insulin	768	79.79947916666667	115.24400235133803	0	846
BMI	768	31.992578124999977	7.884160320375441	0.0	67.1
DiabetesPedigreeFunction	768	0.4718763020833327	0.331328595012775	0.078	2.42
Age	768	33.240885416666664	11.760231540678689	21	81
Outcome	768	0.3489583333333333	0.476951377242799	0	1

Tabla 4.1. Estadísticos básicos del conjunto de datos.

De este modo, en la tabla 4.1. se observa que no existe censura en ninguna de las variables del conjunto, ya que, de ser así, el número de instancias almacenadas en alguna de ellas sería menor al del resto. Por otro lado, a simple vista se observa que el valor mínimo de las variables *"Glucose", "BloodPressure"*, *"SkinThickness", "Insulin"* y *"BMI"* es el cero. Esto es imposible, al tratarse de variables relacionadas con la salud que tienen establecido determinados valores mínimos para que el individuo se encuentre sano.

En el conjunto de datos existen mujeres que nunca han estado embarazadas y otras que, como máximo, han estado embarazadas 17 veces. La media de embarazos entre las mujeres recogidas en el conjunto de datos es de 3.84 embarazos aproximadamente. Mientras que la desviación típica es de 3.37 embarazos aproximadamente.

Por otro lado, el valor medio de la concentración de glucosa en plasma después de 2 horas de realización del Test de Tolerancia a la Glucosa es de 120.89 mg/dl. Con una desviación típica para los valores de esta variable de 31.97 mg/dl. El valor máximo de esta variable es de 199 mg/dl.

La tensión arterial diastólica media de las mujeres que componen la muestra es de 69.10 mm/Hg, valor que se encuentra dentro de los valores normales de esta variable y tiene una desviación típica de 19.35 mm/Hg aproximadamente. El valor máximo es de 122 mm/Hg.

El grosor medio de la piel del tríceps es de 20.54 mm aproximadamente, con una desviación típica de 15.95 mm. El valor máximo que alcanza esta variable es de 99 mm.

La concentración de insulina sérica a las 2 horas de una prueba de tolerancia a la glucosa es de 79.79 μ U/ml aproximadamente, con una desviación típica de, aproximadamente, 155.24 μU/ml. El valor máximo que toma esta variable es de 846 μU/ml. Por lo que, teniendo en cuenta lo anterior, se puede decir que los datos se encuentran muy dispersos.

El Índice de Masa Corporal (IMC) o BMI (en inglés) medio de las mujeres que componen el conjunto de datos es de 31.99, un valor que se encuadra dentro de los valores establecidos para una condición de obesidad moderada (IMC = 30-34.9). La desviación típica de esta variable es de 7.88 y el valor máximo registrado para ella es de 67.1. Este último se trata de un valor extremo, relacionado con una situación de obesidad muy severa o mórbida, de acuerdo con los valores establecidos por la OMS (s.f.).

La variable que recoge los antecedentes familiares de diabetes, toma un valor medio de 0.47 familiares por cada caso con diabetes y posee una desviación típica de 0.078 familiares por cada caso con diabetes. Además, esta variable toma valores entre 0.078 y 2.42 familiares por cada caso con diabetes.

Finalmente, la edad media de las mujeres que componen el conjunto de datos es de 33.24 años. Con una desviación típica de 11.76 años, las mujeres tienen una edad comprendida entre los 21 y los 81 años.

Por otro lado, si nos detenemos a observar el número de mujeres que compone cada grupo de ellas. El grupo de mujeres diabéticas que compone el conjunto de datos, está integrado por 268 mujeres (aproximadamente el 65% del total). Mientras que el grupo de mujeres no diabéticas está compuesto por 500 mujeres (aproximadamente el 65% del total de mujeres). Por lo que, en la fase de preprocesado de los datos, se tendrá que corregir el desbalanceo existente en las clases de la variable “Outcome”. Este hecho puede observarse mejor en el siguiente gráfico de barras.

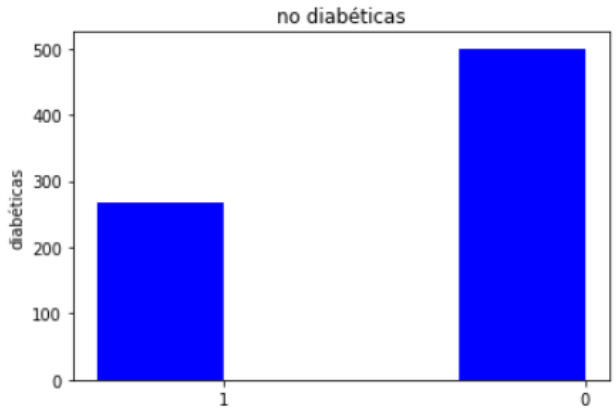

Gráfico 4.1. Número de mujeres que pertenecen a cada clase de la variable de salida.

Ahora que los datos son más familiares, se puede pasar a la búsqueda de valores nulos y de valores perdidos en los datos. Para ello, en primer lugar, se hace un recuento numérico de ambos tipos de datos. Posteriormente, se representan todas las variables mediante gráficos de caja en busca de valores atípicos.

Para realizar este recuento numérico detallado para cada variable del conjunto de datos se ha empleado el siguiente código, extraído de la publicación en el foro online "*How to find count of Null and Nan values for each column in a Pyspark dataframe efficiently*?" (user818327, 2017).

```
print('RECUENTO DE VALORES NULOS:')
print(df.select([count(when(isnan(c), c)).alias(c) for c in df.columns]).show())

print('\nRECUENTO DE VALORES NOT A NUMBER (NaN):')
df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).show()
```

Como puede observarse a continuación en las Tablas 4.1. y 4.2., ninguna de las variables que componen el conjunto de datos contiene valores nulos o perdidos. Aunque, como se ha podido ver cuando se han mostrado los estadísticos básicos, existían variables que tomaban el valor 0. Este valor, se puede tratar como un valor imposible dentro de este conjunto de datos. Esto se debe a que, por ejemplo, una hormona como la insulina, nunca puede tomar el valor cero en un individuo.

RECUENTO DE VALORES NULOS:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0	0	0	0	0	0	0	0

Tabla 4.1. Recuento de valores nulos en el conjunto de datos.

RECUENTO DE VALORES NOT A NUMBER (NaN):

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0	0	0	0	0	0	0	0

Tabla 4.2. Recuento de valores nulos en el conjunto de datos.

Para poder observar mejor este fenómeno, representamos mediante un gráfico de caja las distintas variables que contienen como valor mínimo el cero y no deberían contenerlo. Este gráfico también servirá para observar en torno a qué valores se concentran los datos de estas variables.

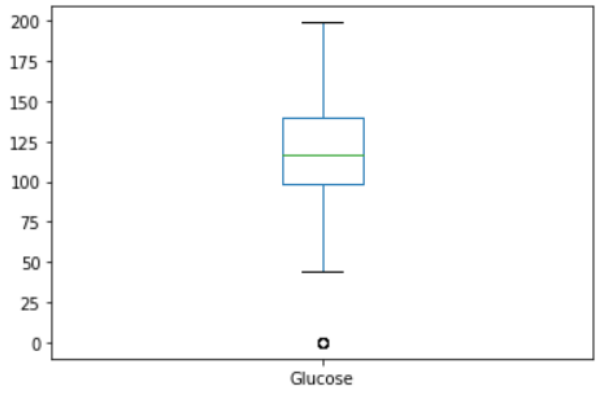

Gráfico 4.2. Gráfico de caja de la distribución de los valores de la variable *"Glucose"*.

En primer lugar, se observa en el Gráfico 4.2., que la variable *"Glucose"* concentra sus valores entre 100 mg/dl y aproximadamente 150 mg/dl. Además, se encuentran más o menos centrados respecto a los posibles valores que puede tomar la variable. Se observa perfectamente que algunas instancias como valor imposible el cero.

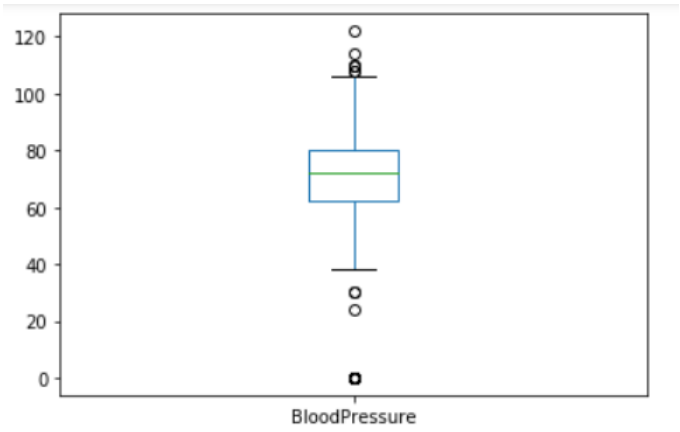

Gráfico 4.3. Gráfico de caja de la distribución de los valores de la variable *"BloodPressure"*

Por otro lado, en el Gráfico 4.3. se observa como la tensión arterial diastólica, concentra la distribución de los datos en torno a los valores 60 a 80 mm/Hg, es decir, en torno a los valores normales o saludables que debe tomar esta variable. Existen valores extremos que superan los 100 mm/Hg, que pueden deberse a que en los datos se incluyen a mujeres de avanzada edad y que pueden padecer de hipertensión.

Existen valores atípicos por debajo del valor que se considera normal para la tensión arterial diastólica (60 mm/Hg) y que lleva a pensar que algunas mujeres padecen de hipotensión. De nuevo, se observa que algunas instancias toman el valor 0 para esta variable.

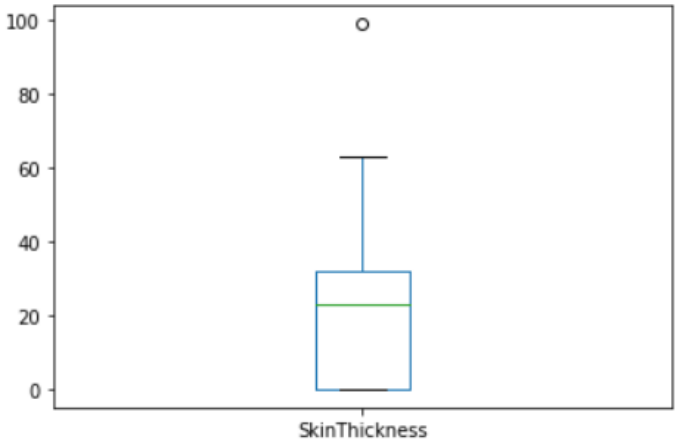

Gráfico 4.4. Gráfico de caja de la distribución de los valores de la variable *"SkinThickness"*

En el caso del grosor de la piel, en el Gráfico 4.4. se observa como los datos son asimétricos hacia los valores inferiores (hacia la derecha) y se concentran entre el valor 0 mm y el valor 25 mm. Existe un valor atípico, que es de en torno a 100 mm.

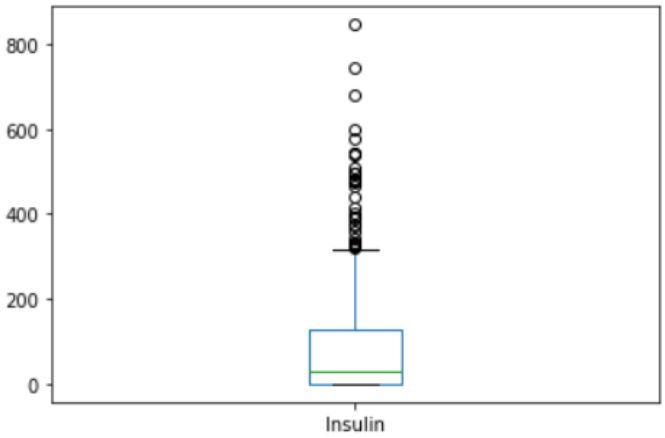

Gráfico 4.5. Gráfico de caja de la distribución de los valores de la variable *"Insulin"*

La variable que concentración de insulina concentra los valores entre 0 y 150 μU/ml aproximadamente. Existiendo valores extremos por encima de aproximadamente 350 μU/ml. Por otro lado, los datos son asimétricos hacia los valores más bajos. Para corregirlo, cuando preprocesen los datos, se deberá eliminar el valor cero.

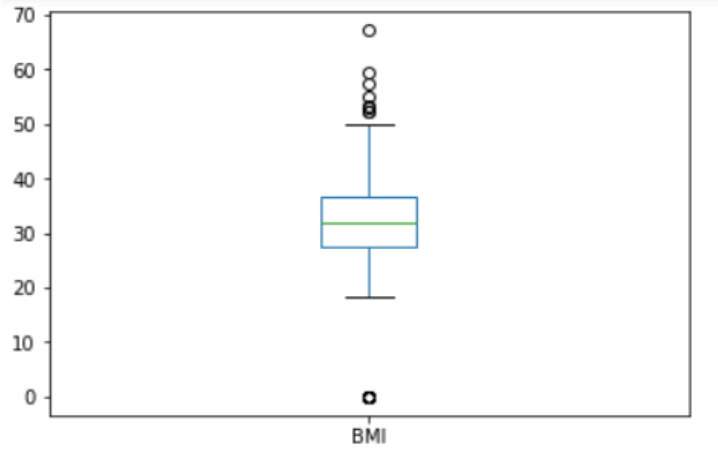

Gráfico 4.6. Gráfico de caja de la distribución de los valores de la variable *"BloodPressure"*

Finalmente, los valores del IMC se concentran entre los 25 y 35 aproximadamente, como se puede observar en el Gráfico 4.6. Algo superiores a los valores saludables recomendados para esta variable según la OMS (s.f.) (18.5 -54.9). Se observan valores extremos por encima del valor 50 (valor que refleja una situación de obesidad mórbida). Pese a ello, se observa que los datos están más o menos centrados. aunque nos centraremos en eliminar el valor cero.

Todos estos gráficos se han obtenido tras aplicar la siguiente sentencia de código para el caso particular de cada una de las distintas variables que se han señalado anteriormente.

```
preg=df.select('Pregnancies').toPandas().plot(kind='box')
```

Para terminar con este análisis, mostramos el valor de la matriz de correlación para las variables que componen el conjunto de datos. Esta matriz se ha construido a partir del siguiente trozo de código, elaborado por Madaka (2019).

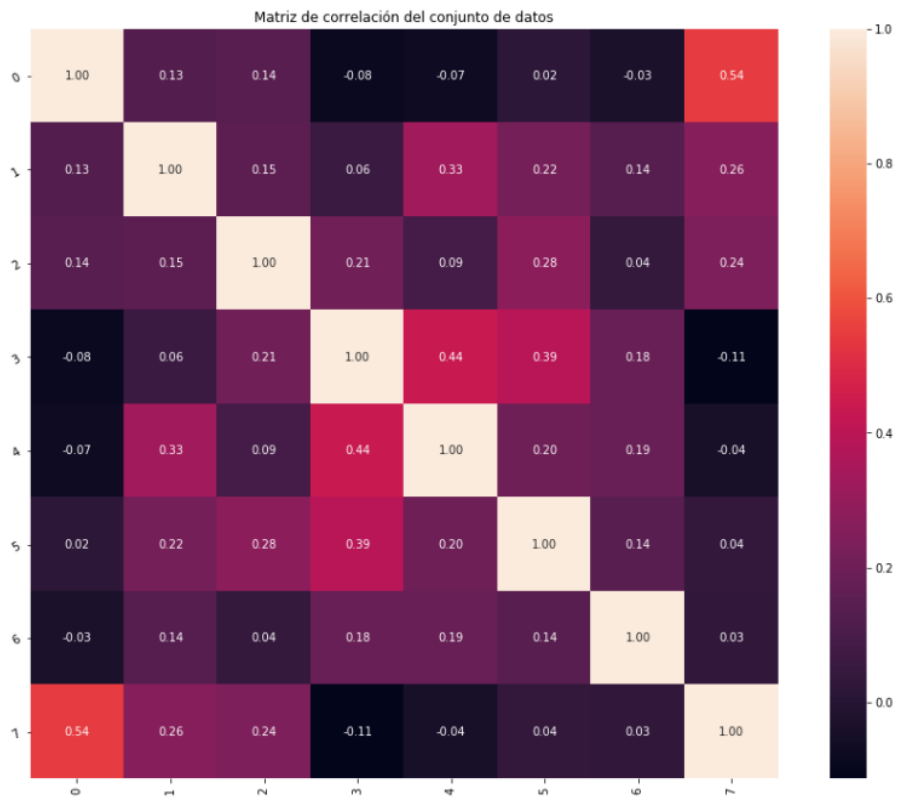

Figura 4.2. Matriz de correlación entre variables del conjunto de datos.

Como puede observarse en la figura superior, no existen valores para la correlación entre pares de variables que sean significativamente elevados.

4.3 Proceso de KDD en el caso de uso.

En los siguientes subapartados se mostrará el desarrollo en Apache Spark del proceso de KDD. Para ejemplificarlo, se toma el conjunto de datos descrito en el apartado anterior.

4.3.1 Fase de preprocesamiento de los datos

Durante esta fase del proceso de KDD, se corregirán los problemas relativos a la existencia de variables con instancias que toman el valor cero (que supone un valor atípico en muchas variables). Además de los relativos a la existencia de clases desbalanceadas y a la normalización de los datos.

4.3.1.1 Imputación de valores atípicos

En primer lugar, para las variables que toman valores imposibles iguales a cero, los convertiremos en valores perdidos. Así, se podrá cuantificar el porcentaje de ellos existentes en el dominio de la variable en cuestión. Para, posteriormente, limpiar estos datos empleando la imputación de estos valores mediante la media de los valores de la variable.

Para convertir, representar y limpiar estos datos se emplearán distintos trozos de código elaborados por Lugat (2020).

Para la conversión de estos datos, se convertirá el conjunto de datos original en un data frame de la librería *Pandas* y se reemplazarán de la siguiente manera (Lugar, 2020):

```
df_pandas= df.toPandas()

df_pandas[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df_pandas[['Glucose',
                'BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)
```

A continuación, se diseña la función “missing_plot” (Lugat, 2020), que ayudará a representar mediante un gráfico de barras el porcentaje de valores perdidos existente en cada variable del conjunto de datos.

```
# Definimos la función que representará la gráfica del porcentaje de valores NaN en cada variable
def missing_plot(dataset, key) :
    null_feat = pd.DataFrame(len(dataset[key]) - dataset.isnull().sum(), columns = ['Count'])
    percentage_null = pd.DataFrame((len(dataset[key]) - (len(dataset[key]) -
                      dataset.isnull().sum()))/len(dataset[key])*100, columns = ['Count'])
    percentage_null = percentage_null.round(2)

    trace = go.Bar(x = null_feat.index, y = null_feat['Count'] ,opacity = 0.8, text = percentage_null['Count'],
                   textposition = 'auto',marker=dict(color = '#7EC0EE', line=dict(color='#000000',width=1.5)))

    layout = dict(title =  "Missing Values (count & %)")

    fig = dict(data = [trace], layout=layout)
    py.iplot(fig)

# Llamamos a la función para representar el gráfico de barras
missing_plot(df_pandas, 'Outcome')
```

El gráfico de barras que devuelve es el que se muestra a continuación.

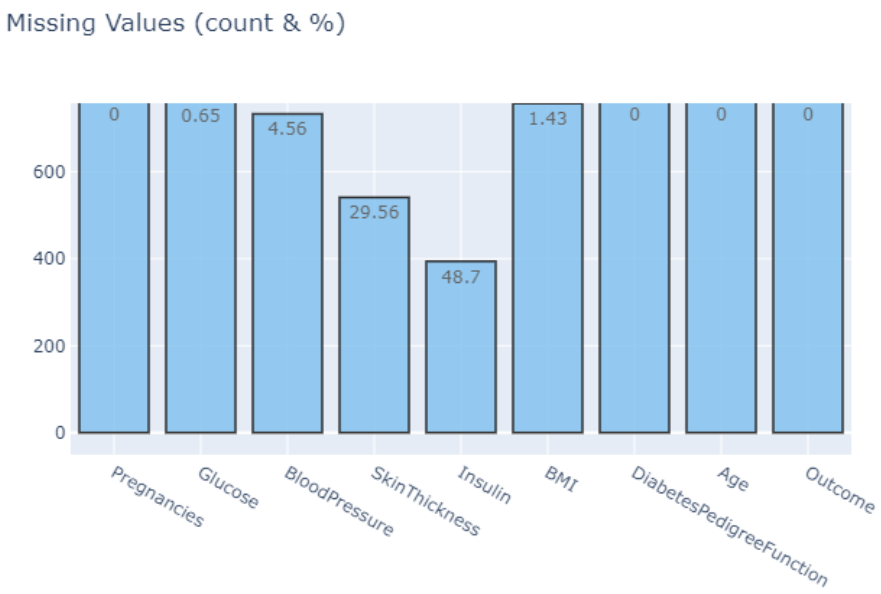

Gráfico 4.7. Porcentaje de valores perdidos en las distintas variables del conjunto de datos.

Se puede ver en el Gráfico 4.7. como la variable que registra un mayor porcentaje de valores perdidos es *"Insuline"*. El 48.7% de las instancias recogidas en la variable toma valores perdidos. Le sigue *"SkinThickness"*, para la cual, el 29.56% de las instancias toma valores perdidos. Seguida de *"BloodPressure"*, para la cual, el 4.56% de las instancias toma valores perdidos. Finalmente, *"Glucose"* y *"BMI"* tienen porcentajes residuales de estos valores entre sus instancias (0.65% y 1.43%, respectivamente).

A continuación, se definirá una función que calcule la media para aquellos valores no nulos de las distintas variables. Este será el estadístico que se emplee a la hora de imputar los valores perdidos reconocidos anteriormente. Además, servirá para conocer la media de cada variable para las mujeres que son diabéticas y para las que no lo son.

Las funciones empleadas para calcular la media de las variables en base a la variable de salida *"Outcome"*, representar la función de distribución de las distintas variables y reemplazar los valores perdidos se muestran en la Sección Apéndice de este trabajo. Los códigos relativos a estas funciones y técnicas han sido tomados de Lugat (2020).

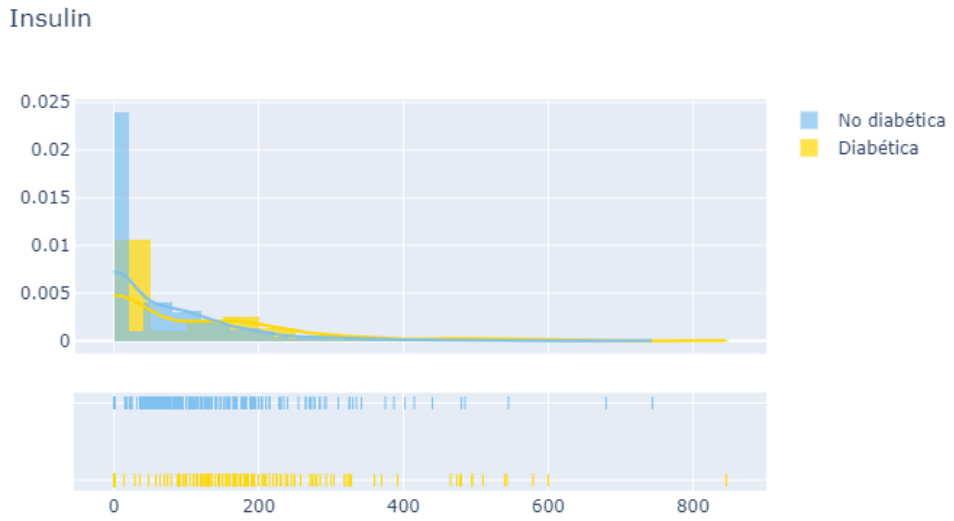

Gráfico 4.8. Función de distribución de la concentración de insulina en ambos grupos de mujeres.

En el Gráfico 4.8., se observa que la función de distribución para la concentración de insulina sérica en las mujeres diabéticas (en azul) es algo superior a la de las mujeres no diabéticas (en amarillo). Asimismo, los valores para la concentración de insulina en las mujeres no diabéticas se concentran más a la izquierda (toman valores más bajos) y están más concentrados. Mientras que los valores de esta variable se encuentran algo más dispersos y algo más desplazados hacia la derecha (toman valores algo más elevados).

Media de la concentración de insulina
según si la mujer es o no diabética (mU/ml)

	Outcome	Insulin
0	0	102.5
1	1	169.5

Tabla 4.3. Media de la concentración de insulina para cada grupo de mujeres.

Lo comentado anteriormente, se puede observar en la tabla 4.3. Muestra la media para la concentración de insulina sérica cuando la mujer es diabética y cuando no lo es. De este modo, las mujeres no diabéticas poseen una concentración media de insulina sérica tras la realización de la prueba de 102.5 mU/ml.

Mientras que la concentración media de insulina sérica tras la realización de la prueba en las mujeres diabéticas es de 169.5 mU/ml.

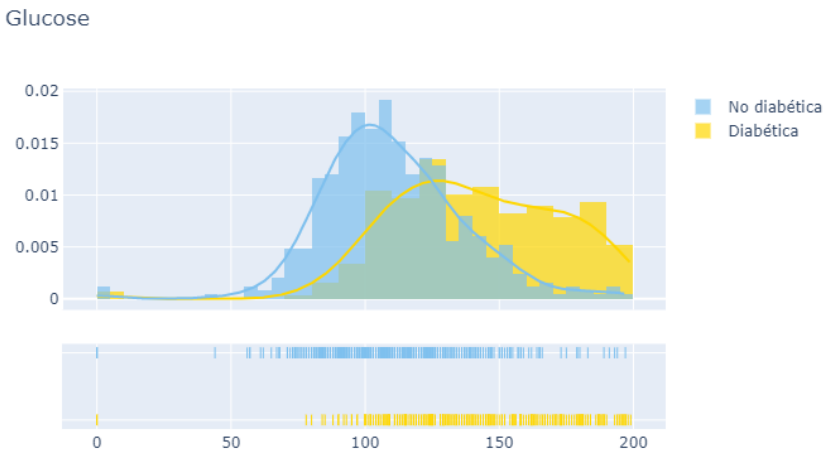

Gráfico 4.9. Función de distribución de la concentración de glucosa en ambos grupos de mujeres.

En el Gráfico 4.9. se puede observar como la función de distribución de la concentración de glucosa plasmática en mujeres no diabéticas se distribuye parecida a una Normal, aunque ligeramente desplazada a la derecha. La función de distribución para esta variable en mujeres diabéticas se encuentra aún más desplazada a la derecha (toma valores superiores) y no sigue ninguna distribución conocida.

Esta diferencia en los valores para la concentración de glucosa plasmática se puede observar mejor en la Tabla 4.4.

Media de la concentración de glucosa plasmática
según si la mujer es o no diabética (mg/dl)

	Outcome	Glucose
0	0	107
1	1	140

Tabla 4.4. Media de la concentración de glucosa para cada grupo de mujeres.

Las mujeres que no padecen diabetes poseen una concentración media de glucosa plasmática igual a 107 mg/dl. Mientras que las mujeres que sí la padecen, poseen una concentración media igual a

140 mg/dl. Ambos valores tienen sentido, ya que, por definición, las mujeres diabéticas deben poseer valores superiores de concentración de glucosa plasmática.

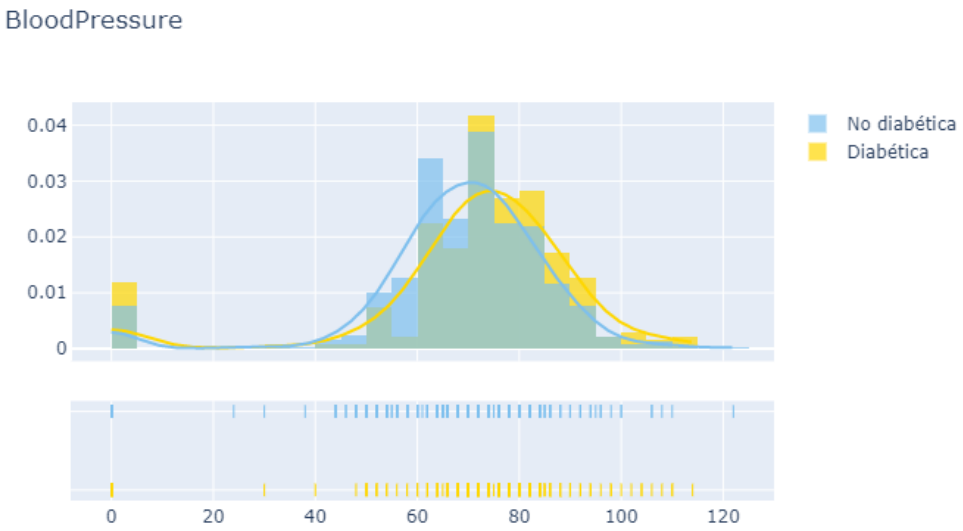

Gráfico 4.10. Función de distribución de la tensión arterial diastólica en ambos grupos de mujeres.

La tensión arterial diastólica a partir del valor 20 mmHG en adelante (aproximadamente), se distribuye como una Normal. Esto sucede para ambas funciones de distribución de mujeres diabéticas y no diabéticas.

No obstante, es cierto que la función de distribución correspondiente a las mujeres diabéticas (en amarillo) alcanza un máximo menor al de la función correspondiente a las mujeres no diabéticas (en azul). Además, la función de distribución de la tensión arterial en mujeres diabéticas se encuentra desplazada más a la derecha que la correspondiente a las mujeres no diabéticas.

Finalmente, es muy notoria la concentración de los valores para ambas funciones de distribución en torno al cero. Motivo por el que se corrige este aspecto mediante la imputación de estos valores por la media de la variable, que se muestra en la siguiente tabla.

Media de la tensión arterial diastólica
según si la mujer es o no diabética (mmHg)

	Outcome	BloodPressure
0	0	70.0
1	1	74.5

Tabla 4.5. Media de la tensión arterial diastólica para cada grupo de mujeres.

En la Tabla 4.5., se muestran los valores de la tensión arterial diastólica media de las mujeres diabéticas y no diabéticas (mmHG). De este modo, las mujeres no diabéticas tienen una tensión arterial diastólica media de 70 mmHG. Este valor es más elevado en el caso de las mujeres diabéticas, su tensión arterial diastólica media es de 74.5 mmHG.

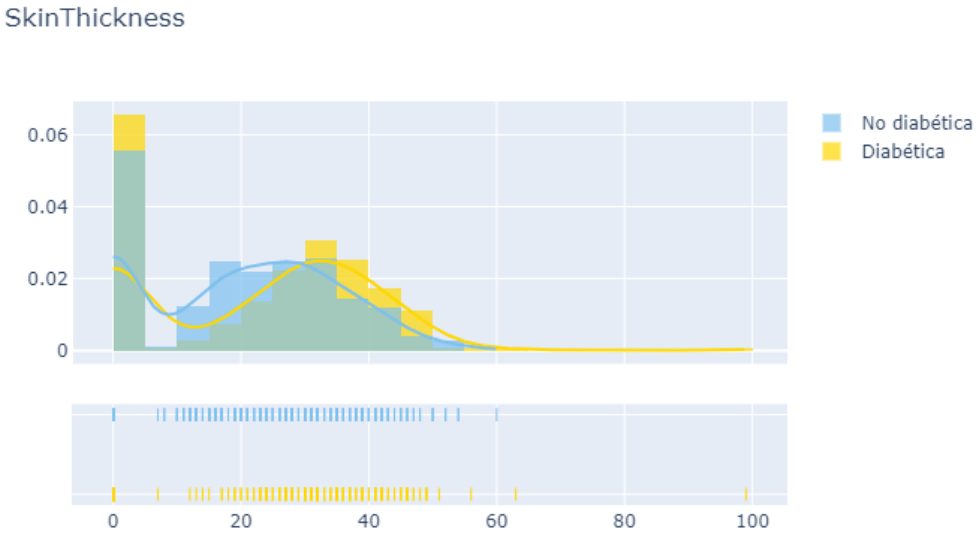

Gráfico 4.11. Función de distribución del grosor de la piel del tríceps en ambos grupos de mujeres.

En relación al grosor de la piel del tríceps de las mujeres que componen el conjunto de datos, llama mucho la atención la concentración de valores en torno al cero (como se ha visto antes el 29.56% de las instancias de la variable *"SkinThickness"* tomaban el valor cero).

Sin tener en cuenta estos valores, ambas funciones de distribución de grosor de la piel del tríceps siguen una distribución similar a la Normal. En el caso de la función de distribución para mujeres no diabéticas se vuelve más plana cuando se acerca al centro de la distribución.

Media del grosor de la piel según si la mujer es o no diabética (mm)

	Outcome	SkinThickness
0	0	27
1	1	32

Tabla 4.6. Media del grosor de la piel del tríceps para cada grupo de mujeres.

En la tabla 4.6. se confirma un hecho que se puede observar en la gráfica que muestra las funciones de distribución del grosor de la piel para mujeres diabéticas y no diabéticas. Este hecho es que ambas funciones alcanzan su máximo en valores muy cercanos.

Estos valores coinciden con la media de esta variable para cada grupo de mujeres. De este modo, las mujeres diabéticas tienen un grosor medio de la piel del tríceps de 32 mm, frente a los 27 mm de grosor que poseen las mujeres no diabéticas.

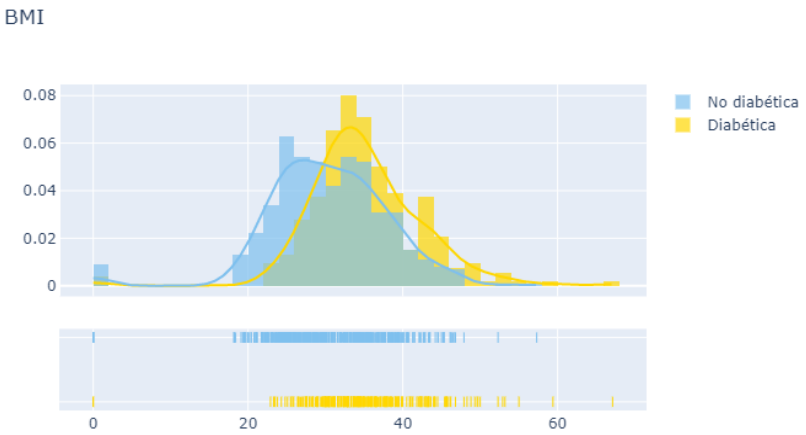

Gráfico 4.12. Función de distribución del Índice de Masa Corporal en ambos grupos de mujeres.

Se pasa, finalmente, a hablar de las funciones de distribución del IMC para ambos grupos de mujeres. La relativa al IMC del grupo de mujeres diabéticas (en amarillo) es más apuntada y más regular que la relativa al grupo de mujeres no diabéticas. Para este último grupo de mujeres se observa una función más achatada, más irregular, descentrada y desplazada a la izquierda.

```
Media del Índice de Masa Corporal según si la mujer es o no diabética

   Outcome   BMI
0        0  30.1
1        1  34.3

------------------------------
```

Tabla 4.7. Media del Índice de Masa Corporal para cada grupo de mujeres.

Los valores de la media para ambos grupos de mujeres respectivamente, servirán, como se ha comentado, para imputar los valores iguales a ceros convertidos en valores perdidos. Así, el IMC medio de las mujeres no diabéticas es de 30.1, frente al IMC medio de 34.3 de las mujeres no diabéticas.

De acuerdo con el criterio de la Organización Mundial de la Salud (s.f.), estos valores medios de IMC reflejan que ambos grupos de mujeres, en media, se encuentran en situación de obesidad. En el caso de las mujeres no diabéticas, de obesidad moderada, y en el caso de las mujeres diabéticas, de obesidad severa.

Para finalizar con la eliminación e imputación de valores imposibles, se muestran a continuación los estadísticos básicos para el conjunto de datos corregido. Esto nos sirve para comprobar que no quedan valores perdidos ni ceros en las variables que se han corregido con el procedimiento anterior.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.00000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.677083	72.378906	27.43099	141.753906	32.433893	0.471876	33.240885	0.348958
std	3.369578	30.464161	12.104431	9.32146	89.100847	6.880657	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.00000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	21.00000	102.500000	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	27.00000	102.500000	32.050000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.00000	169.500000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.00000	846.000000	67.100000	2.420000	81.000000	1.000000

Tabla 4.8. Estadísticos básicos del conjunto de datos tras imputar los valores perdidos.

Como se puede observar en la Tabla 4.8., no aparece el cero como valor mínimo para las variables que hemos corregido. Así, el valor mínimo para la concentración de glucosa pasa a ser de 44 ml/dl, el valor para la tensión arterial diastólica es de 24 mmHG y el grosor mínimo de la piel existente

en el conjunto de datos es de 7 mm. Asimismo, el valor mínimo de la concentración de insulina es 14 mU/ml y el valor mínimo del IMC es de 18.20.

4.3.1.2 Corrección de las clases no balanceadas

Ahora que ya se ha corregido el problema relativo a las instancias que tomaban valor cero para variables que no debían tomarlo. Se pasa a corregir el problema de desbalanceo entre clases de la variable de salida.

En un principio, el grupo de mujeres diabéticas que compone el conjunto de datos está integrado por 268 (aproximadamente el 65% del total de mujeres) mujeres. Mientras que el grupo de mujeres no diabéticas está compuesto por 500 mujeres (aproximadamente el 65% del total de mujeres).

Para realizar la corrección del balanceado de clases se ha empleado la función *SMOTE*. Esta función generará nuevas instancias para la clase minoritaria (mujeres diabéticas), a partir de las instancias existentes. De este modo, se obtendrán dos grupos de clases que supongan, respectivamente, el 50% de las instancias.

```
Balance de clases del dataset:
Numero de datos de no_enfermos -0-: 500 (65.10416666666666%)
Numero de datos de enfermos -1-: 268 (34.89583333333333%)

Balance de clases del dataset - NUEVO BALANCE DE CLASES:
Numero de datos de no diabéticos (0): 500 (50.0%)
Numero de datos de diabéticos (1): 500 (50.0%)
```

Tabla 4.9. Balanceo de clases antes y después de la corrección.

El código empleado para balancear las clases se encuentra detallado en la Sección Apéndice de este trabajo y ha sido extraído de Madaka (2019).

4.3.1.3 Normalización de los datos

A continuación, se pasa a normalizar las variables del conjunto de datos. El código empleado ha sido extraído de Madaka (2019) y se encuentra detallado en el Apéndice de este trabajo.

La normalización consiste en transformar los valores de las variables del conjunto de datos para que tomen valores entre 0 y 1. Se procede tomando el menor valor de cada variable y

transformándolo en cero. Mientras que el mayor valor se transforma en 1. El resto de valores intermedios se transforman mediante una regla de tres para que tomen valores entre 0 y 1.

En la Tabla 4.10., que se muestra a continuación, quedan reflejados los valores que toman las distintas variables del conjunto de datos sin normalizar. Sólo se muestran las 5 primeras filas, debido a que el conjunto de datos es muy amplio y es complicado mostrar todas.

```
+-----------------------------------------+
|features                                 |
+-----------------------------------------+
|[6.0,148.0,72.0,35.0,169.5,33.6,0.627,50.0]|
|[1.0,85.0,66.0,29.0,102.5,26.6,0.351,31.0] |
|[8.0,183.0,64.0,32.0,169.5,23.3,0.672,32.0]|
|[1.0,89.0,66.0,23.0,94.0,28.1,0.167,21.0]  |
|[0.0,137.0,40.0,35.0,168.0,43.1,2.288,33.0]|
+-----------------------------------------+
only showing top 5 rows
```

Tabla 4.10. Conjunto de datos sin normalizar.

La siguiente tabla recoge los resultados de aplicar el código señalado anteriormente para normalizar los valores de las distintas variables.

	features_Scaled
0	[0.35294117647058826, 0.6709677419354839, 0.48...
1	[0.058823529411764705, 0.2645161290322581, 0.4...
2	[0.47058823529411764, 0.896774193548387, 0.408...
3	[0.058823529411764705, 0.2903225806451613, 0.4...
4	[0.0, 0.6, 0.16326530612244897, 0.304347826086...

Tabla 4.11. Conjunto de datos normalizado.

Como se puede observar, todas toman valores entre 0 y 1 y los valores intermedios son continuos. Al igual que en el caso anterior, sólo se muestran las cinco primeras filas y, en este caso, algunos de los valores de las columnas, debido a que si se muestran todas, se colapsa la tabla.

4.3.1.4 Selección de características

Para terminar con la fase de preprocesamiento de los datos, se ha realizado una selección de características sobre el conjunto de características. El código detallado de la esta técnica, tomado de Madaka (2019), se muestra en el Apéndice de este trabajo.

Para seleccionar las características del conjunto de datos, se ha utilizado la función *ChiSqSelector*, disponible en la librería *pyspark.ml.feature.* Esta función emplea un contraste Chi-Cuadrado para seleccionar las mejores características (Apache Spark 2.2.0, s. f.), actuando este contraste como medida de entropía entre variables. Los resultados que genera su aplicación son los siguientes:

ChiSqSelector output with ---[[6.0,148.0,72.0,35.0,169.5,33.6,0.627,50.0]]--- features selected

4.3.2 Fase de Minería de Datos

Al concluir la fase anterior, se obtiene el conjunto de datos limpio y transformado. Por lo que estará listo para aplicar los algoritmos de clasificación que se describen en los siguientes subapartados.

La decisión de aplicar algoritmos de clasificación se basa en la naturaleza de la variable de salida *"Outcome"* del conjunto de datos. Esta variable distingue dos clases dentro del espacio de características: mujeres diabéticas y no diabéticas. Para ello, emplea dos valores, el 0 (mujeres no diabéticas) y el 1 (mujeres diabéticas), convirtiéndose en una variable binaria.

Los algoritmos aplicados son el Clasificador Bayesiano o *Naïve Bayes*, la máquina de vector soporte lineal *Linear Support Vector Classification* (LSVC) y el algoritmo *Random Forest*.

Finalmente, antes de implementar los distintos algoritmos, se debe dividir el conjunto de datos original en dos subconjuntos. El primero de ellos servirá como conjunto de entrenamiento y supondrá el 75% de las instancias. Mientras que el segundo, servirá para probar el modelo y contendrá el 25% de instancias restante.

Esta división se ha realizado mediante la función *randomSplit* de *pypsark*, que permite generar los dos subconjuntos de datos seleccionando de forma aleatoria las instancias que pertenecen a cada uno.

4.3.2.1 Implementación del Clasificador Naïve Bayes

Para implementar este clasificador se ha empleado el código expuesto en la documentación de Apache Spark relativa a Clasificación y Regresión de la librería MLlib de Spark. Por lo tanto, será necesario emplear las librerías *pyspark.ml.classification y pyspark.ml.evaluaation.* En concreto,

importaremos las funciones *NaiveBayes* y *MulticlassClassificationEvaluator*, Apache Spark. (s. f.-b).

La función *NaiveBayes* será la primera que se emplee y servirá para definir el clasificador a implementar.

```
10 # Creamos el modelo de entrenamiento y fijamos las columnas
11 nb = NaiveBayes(featuresCol="selectedFeatures", labelCol="Outcome")
12
```

En la sentencia de código superior se fija la columna que contiene las variables que se emplearán para explicar el modelo. Las instancias que contienen están corregidas mediante todas las técnicas descritas en la fase anterior de preprocesamiento de los datos. Esta columna es *"selectedFeatures"* y es de tipo vector, contiene en una columna todos los valores de todas las variables explicativas del conjunto de datos.

Por otro lado, también se fija la variable de salida *"Outcome"*. Esta determina los valores de las clases y es la que se busca predecir.

```
8  # Entrenamos el modelo
9  modelNB = nb.fit(train)
10 predictionsNBtrain = modelNB.transform(train)
11
12
13 # PRUEBA DEL MODELO
14 predictionsNBtest = modelNB.transform(test)
15
16 print('Valores de la predicción del Clasificador Naïve Bayes para el conjunto de test')
17 predictionsNBtest.select('Outcome', 'Prediction', 'probability').show()
18
```

A continuación, se emplea el trozo de código superior para entrenar el modelo, primero, y probarlo, después.

Durante la fase de entrenamiento, en la línea 9 se ajusta el clasificador definido anteriormente al conjunto de entrenamiento. Esto se realiza mediante la función *fit* aplicada al conjunto de datos contenido en la variable *"train"* del programa. Para luego, realizar las predicciones de la variable *"Outcome"* sobre este conjunto de datos de entrenamiento. Esto último se realiza mediante la función *transform* aplicada al conjunto contenido en la variable *"train"* del programa.

Posteriormente, se prueba el modelo entrenado (contenido en la variable *"modelNB"*) para comprobar que generaliza bien. Esto se realiza mediante la función *transform* aplicada al conjunto de datos contenido en la variable *"test"* del programa.

Finalmente, para cerrar la fase de prueba, se muestran las predicciones (función *show()*) sobre el conjunto de prueba (contenido en la variable *"test"*). Para ello, se deben seleccionar las columnas *"Outcome", "Prediction"* y *"probability"* mediante la función *select.*

La columna *"Outcome"* contiene los valores reales para el conjunto de prueba de la variable del mismo nombre. La columna *"Prediction"* contiene los valores de la predicción para la variable de salida del modelo. Por último, la variable *"probability"* contiene las distintas probabilidades de que una instancia pertenezca a una determinada clase.

La siguiente tarea que se debe desempeñar es la evaluación del modelo. Para conocer como de bueno es el modelo se empleará la función *MulticlassClassificationEvaluator*. Esta función fija la columna que contiene variable de salida del modelo (*"Outcome"*) y la columna que contiene las predicciones (*"predicition"*). Además de invocar a la métrica a utilizar para evaluar el modelo, en este caso el *accuracy* o precisión.

Esta métrica compara el valor predicho para las instancias de la variable de salida con su valor real, expresado en tanto porciento.

```
#Evaluación del modelo
evaluatorNB = MulticlassClassificationEvaluator(labelCol="Outcome", predictionCol="prediction",
                                                metricName="accuracy")
# Bondad del ajuste
print('Bondad del ajuste del Clasificador Naïve Bayes')
print('..............................................')
accuracyNBtrain = evaluatorNB.evaluate(predictionsNBtrain)
print("Accuracy del conjunto de entrenamiento  = " + str(accuracyNBtrain))

accuracyNBtest = evaluatorNB.evaluate(predictionsNBtest)
print("Accuracy del conjunto de test  = " + str(accuracyNBtest))
print('..............................................\n.')
```

Seguidamente, se hace uso de la función *evaluate* para evaluar la predicción obtenida por el modelo a partir de la métrica anterior. De esta forma, se conoce el valor del *accuracy* para el conjunto de entrenamiento (almacenado en la variable *"accuracyNBtrain"*) y de prueba (almacenado en la variable *"accuracyNBtest"*).

Finalmente, se ha querido introducir también una matriz de confusión definida mediante la función *plot_confusion_matrix* (Scikit-learn, s.f.). Esta función ha sido diseñada previamente y su código se encuentra detallado en el Apéndice de este trabajo.

```
# MATRIZ DE CONFUSIÓN
y_trueNB = predictionsNBtest.select(['Outcome']).collect()
y_predNB = predictionsNBtest.select(['prediction']).collect()
cnf_matrixNB = confusion_matrix(y_trueNB, y_predNB,labels=[0,1])

plt.figure()
plot_confusion_matrix(cnf_matrixNB, classes=[0,1],title='Matriz de confusion Modelo Naïve Bayes\n')
plt.show()
```

Esta matriz de confusión ayudará a conocer la tasa de acierto y error de la predicción del modelo. Además del número de instancias correctamente o incorrectamente clasificadas por el Clasificador Bayesiano.

4.3.2.2 Implementación del Clasificador Random Forest

El código de referencia para implementar el clasificador Random Forest Classifier, es el expuesto en la documentación de Apache Spark relativa a Clasificación y Regresión de la librería MLlib de Spark para Python, Apache Spark. (s. f.-b).

Para implementarlo en la API de Spark para Python se hace necesario emplear las librerías *pyspark.ml.classification y pyspark.ml.evaluation.* En concreto, previamente se han importado las funciones *RandomForestClassifier* y *MulticlassClassificationEvaluator*, Apache Spark. (s. f.-b).

Al igual que en el caso anterior, la función *RandomForestClassifier* será la primera que se emplee y servirá para definir el clasificador a implementar.

```
# Clasificación con random forest a partir del conjunto de entrenamiento enterior

rf = RandomForestClassifier(numTrees=200, labelCol="Outcome", featuresCol="selectedFeatures").setMaxDepth(10)

# Creamos la Pipeline
pipeline = Pipeline(stages=[rf])

```

En el trozo de código superior se fija la columna que contiene las variables que se emplearán para explicar el modelo. Esta columna es *"selectedFeatures"* y es de tipo vector, contiene en una columna todos los valores de todas las variables explicativas del conjunto de datos. Asimismo, se fija la variable de salida *"Outcome"*. Esta determina los valores de las clases y es la que se busca predecir. También se fija el número de árboles en 200.

En este clasificador deberá crearse una pipeline, que dirige el flujo de trabajo del clasificador. Para ello se emplea la función *Pipeline* de *pypark*, importada previamente.

```
# FASE DE ENTRENAMIENTO DEL MODELO
modeloRF = pipeline.fit(train)

#FASE DE PRUEBA DEL MODELO
# Realizamos las predicciones. Utilizamos para ello el conjunto de datos de test
predictionsRFtest = modeloRF.transform(test)
predictionsRFtrain = modeloRF.transform(train)

# Mostramos las predicciones
predictionsRFtest.select("prediction", "Outcome", "selectedFeatures").show(5)
```

De nuevo, la fase de entrenamiento se inicia ajustando el modelo al conjunto de entrenamiento creado anteriormente. Para, a continuación, pasar a probar el modelo y generar las predicciones para el conjunto de entrenamiento y test. Estas predicciones se almacenan en las variables *"predicionsRFtest"* y *"predicionsRFtrain"*, respectivamente.

Finalmente, se muestran las predicciones para el conjunto de prueba. Para ello se ha seguido el procedimiento detallado en el clasificador anterior (adaptándolo a este caso particular).

```
# Evaluamos el modelo obtenido mediante la métrica 'precision' y la matriz de confusion
evaluatorRF = MulticlassClassificationEvaluator(labelCol="Outcome", predictionCol="prediction", metricName="accuracy")
accuracyRFtest = evaluatorRF.evaluate(predictionsRFtest)

print('Bondad del ajuste del Clasificador Random Forest')
print('.............................................')

print("\n Error conjunto de test Error = %g" % (1.0 - accuracyRFtest))
print(" Accuracy conjunto de test = %g" % (accuracyRFtest))
accuracyRFtrain = evaluatorRF.evaluate(predictionsRFtrain)
print("\n Error conjunto de entrenamiento = %g" % (1.0 - accuracyRFtrain))
print(" Accuracy conjunto de entrenamiento= %g" % (accuracyRFtrain))
print("\n")
```

Finalizadas las dos fases anteriores (entrenamiento y test) se evalúa el modelo mediante la función *MulticlassClassificationEvaluator* y seleccionando la métrica *accuracy*. Esta métrica se aplicará partiendo de las columnas *"Outcome"* y *"prediction"*, al igual que en el caso anterior.

Esta métrica se ha aplicado, a continuación, a la variable que contiene las predicciones para el conjunto de test (*"predictionsRFtest"*) y para el conjunto de entrenamiento (*"predictionsRFtrain"*).

Por último, para terminar con esta tarea, se muestra el valor del error y del accuracy para el conjunto de entrenamiento y test. El error viene dado por la diferencia entre 1 y el valor del accuracy para cada subconjunto en tanto por uno.

La representación de la matriz de confusión para este clasificador será la medida de bondad del ajuste que cierre la implementación del clasificador Random Forest.

```
# MATRIZ DE CONFUSIÓN
# Calculamos la matriz de confusion y la mostramos mediante un grafico
y_true = predictionsRFtest.select(['Outcome']).collect()
y_pred = predictionsRFtest.select(['prediction']).collect()
cnf_matrix = confusion_matrix(y_true, y_pred,labels=[0,1])

plt.figure()
plot_confusion_matrix(cnf_matrix, classes=[0,1],title='Matriz de confusion Random Forest')
plt.show()

# Mostramos un resumen del clasificador que hemos utilizado
rfModel = modeloRF.stages[0]
print(rfModel)
```

Para ello se ha empleado el mismo código que en el caos del Clasificador Bayesiano, adaptado a este clasificador y modificando los nombres de las distintas variables.

4.3.2.3 Implementación del algoritmo Linear Support Vector Classification.

El último algoritmo que se implementará para ejemplificar esta fase del proceso de KDD será Linear Support Vector Classification (LSVC). Al igual que en el caso de los dos algoritmos anteriores, se ha recurrido a la documentación de Apache Spark. Se ha tomado como referencia el código expuesto en la documentación de la librería MLlib para Clasificación y Regresión de de Spark, Apache Spark. (s. f.-a).

Para implementar este clasificador en la API de Spark para Python se importan las librerías *LinearSVC* de *pyspark.ml.classification y MulticlassClassificationEvaluator de pyspark.ml.evaluation*, Apache Spark. (s. f.-b).

```
# IMPLEMENTAMOS EL CLASIFICADOR LSVC
lsvc = LinearSVC(featuresCol="selectedFeatures", labelCol="Outcome", maxIter=20)

# ENTRENAMIENTO DEL MODELO

# Ajustamos el modelo
lsvcModel = lsvc.fit(train)
```

Para implementar este clasificador se procede igual que se ha procedido para los dos clasificadores anteriores. En primer lugar, se llama al clasificador mediante la función *LinearSVC*, se fijan las columnas con las variables explicativas y con la variable de salida y se fija el número máximo de iteraciones en 20. Es un número muy bajo de iteraciones, pero así, nos aseguramos de que no tarde mucho en ejecutar.

A continuación, en la línea de código número 7, se entrena el modelo. Para ello, se ajusta el clasificador anterior a los datos de entrenamiento mediante la función *fit*.

```
# PRUEBA DEL MODELO - LSVC

predictionslsvctest = lsvcModel.transform(test)

print("\n Valores de la predicción LSVC para el conjunto de test")
predictionslsvctest.select('Outcome', 'Prediction').show()

# Imprimimos los valores de los coeficientes y el intercepto
print('_____________________________________________\n')
print("VALORES DE LOS COEFICIENTES E INTERCEPTO MODELO LSVC\n.....................................................")
atrib=dfprep.columns[1:9]

for i in range(len(lsvcModel.coefficients)):
    print(atrib[i],':',lsvcModel.coefficients[i])
print('.....................................................')

#print("VALOR DEL INTERCEPTO MODELO LSVC\n.....................................")
print('Intercepto:',lsvcModel.intercept )
print('_____________________________________________________')
```

Seguidamente, se prueba el modelo aplicando la función *transform* al conjunto de test. Se aplica también al conjunto de entrenamiento para poder calcular el accuracy.

Para mostrar la predicción en base a las instancias de prueba, se emplea las funciones *select* (para seleccionar las variables *"Outcome"* y *"Prediction"*) y *show* (para mostrar los valores por pantalla en formato tabla).

También se muestran los valores de los coeficientes de la función que divide en dos las instancias y el valor de la constante de este modelo.

Finalmente, para evaluar la bondad del ajuste del clasificador se utiliza el accuracy. Esta métrica es necesario invocarla y aplicarla sobre el conjunto de prueba como se ha descrito antes (utilizando las funciones *MulticlassClassificationEvaluator* y *evaluate*, respectivamente).

Al igual que se ha hecho con las predicciones, se aplica también esta métrica al conjunto de entrenamiento para conocer la bondad del ajuste del modelo a estas instancias,

```
print('Bondad de ajuste del Clasificador LSVC')
print('.....................................................')
eval_lsvc = MulticlassClassificationEvaluator(labelCol="Outcome", predictionCol="prediction",
                                              metricName="accuracy")

accuracy_lsvctrain = eval_lsvc.evaluate(predictionslsvctrain)
print("Accuracy conjunto de entrenamiento  = " + str(accuracy_lsvctrain))
print('')
accuracy_lsvctest = eval_lsvc.evaluate(predictionslsvctest)
print("Accuracy conjunto de test  = " + str(accuracy_lsvctest))
print('_____________________________________________________')

# MATRIZ DE CONFUSIÓN
print('\n')
# Calculamos la matriz de confusion y la mostramos mediante un grafico
y_truelsvc = predictionslsvctest.select(['Outcome']).collect()
y_predlsvc = predictionslsvctest.select(['prediction']).collect()
cnf_matrixlsvc = confusion_matrix(y_truelsvc, y_predlsvc,labels=[0,1])

plt.figure()
plot_confusion_matrix(cnf_matrixlsvc, classes=[0,1],title='Matriz de confusion Modelo LSVC')
plt.show()
```

Además del accuracy, se ha empleado también una matriz de confusión. Su representación se ha realizado empleando la misma función y el mismo código que en los casos anteriores. Sólo se han sustituido los valores de las variables para adaptarla al caso de este clasificador.

4.3.3 Interpretación de los resultados

Esta fase cierra el proceso de KDD y sirve para conocer si todo lo realizado anteriormente es correcto. A partir de los resultados obtenidos para los distintos clasificadores, se podrá conocer mejor la realidad de las mujeres que componen este conjunto de datos y las predicciones que arrojan estos algoritmos en base a sus características.

En los siguientes subapartados se comentan estos resultados y la bondad del ajuste de los distintos modelos sin entrar en demasiados detalles, ya que no es el objetivo del presente trabajo y, además, se requeriría un estudio más detallado. La interpretación se limitará a comentar los resultados desde un punto de vista descriptivo para ilustrar la metodología que se detalla, dejando siempre la puerta abierta a futuros trabajos en esta línea.

Para terminar, se hará una breve comparativa de la bondad de ajuste de los distintos modelos para el conjunto de datos empleado.

4.3.3.1 Resultados del Clasificador Naïve Bayes

El Clasificador Naïve Bayes es un algoritmo muy sencillo, basado en la probabilidad de que una determinada instancia pertenezca a una clase. En este caso, se quería conocer la probabilidad de que una mujer padezca diabetes en base a las características consideradas en el conjunto de datos.

Por ello, una vez entrenado y probado el clasificador, se han realizado una serie de predicciones para las mujeres de la muestra. Estas predicciones se muestran a continuación para 20 mujeres del conjunto de datos.

Outcome	Prediction	probability
0.0	0.0	[0.99999934776190...
0.0	0.0	[0.99999955674846...
0.0	0.0	[0.99529022617433...
0.0	1.0	[0.00240569327040...
0.0	0.0	[0.99314751877341...
0.0	0.0	[0.96669958699702...
0.0	1.0	[0.01401388540947...
0.0	0.0	[0.98886070871348...
0.0	0.0	[0.99955648241380...
0.0	1.0	[2.98752782575108...
0.0	0.0	[0.98361795886190...
0.0	1.0	[0.36793360862176...
0.0	0.0	[0.99949022994563...
1.0	1.0	[0.00684710231402...
0.0	0.0	[0.99879433190227...
0.0	0.0	[0.99287157716400...
1.0	1.0	[0.00371817503333...
0.0	1.0	[3.52631107075232...
1.0	1.0	[2.49559318424523...
0.0	0.0	[0.99986069585658...

only showing top 20 rows

Tabla 4.12. Valores de la predicción del Clasificador Naïve Bayes para el conjunto de datos de prueba.

En la columna *"Prediction"* de la tabla superior puede observar que el clasificador falla en la predicción para algunas instancias. Si comparamos estos valores con los reales, recogidos en la columna *"Outcome",* la predicción es correcta para las mujeres que padecen diabetes. En el caso de mujeres no diabéticas, la bondad de la predicción cambia. El clasificador falla al clasificar 5 de las 17 mujeres no diabéticas como diabéticas.

No obstante, hay que tener en cuenta que esto sucede para las 20 mujeres que se muestran en la Tabla 4.12. y que representan una parte muy pequeña del total de mujeres del conjunto. Por lo que, es difícil juzgar la bondad del ajuste del modelo basándonos sólo en los valores de estas predicciones.

Por este motivo, se realiza la evaluación del modelo mediante la métrica *accuracy* de bondad del ajuste y una tabla de confusión. Los resultados para ambos aspectos se muestran a continuación.

```
Bondad del ajuste del Clasificador Naïve Bayes
.........................................
Accuracy del conjunto de entrenamiento  = 0.7647058823529411
Accuracy del conjunto de test  = 0.7769516728624535
.........................................
```

Tabla 4.13. Bondad del ajuste del Clasificador Naïve Bayes.

El Clasificador Naïve Bayes es capaz de clasificar correctamente las instancias de entrenamiento con una precisión del 76.70%. Esta precisión en la clasificación aumenta para las instancias de prueba hasta el 77.69%. Estos resultados, según la literatura para un caso teórico son muy bajos y el clasificador no estaría proporcionando el mejor ajuste. Se necesitarían valores de *accuracy* superiores al 90% para concluir que el ajuste es óptimo.

El problema es que en la realidad, los datos son mucho más heterogéneos y presentan muchos problemas a la hora de ajustar un modelo. En la práctica, valores de *accuracy* entre el 70%-80% se consideran que son muy buenos, e incluso, se aceptan valores inferiores para validar la capacidad de generalizar del modelo.

Por este motivo, al tratarse de datos de mujeres reales, se puede decir que el Clasificador Naïve Bayes es capaz de generalizar adecuadamente a partir de instancias desconocidas.

De todos modos, para validar esta conclusión, se comenta también la matriz de confusión generada por el clasificador.

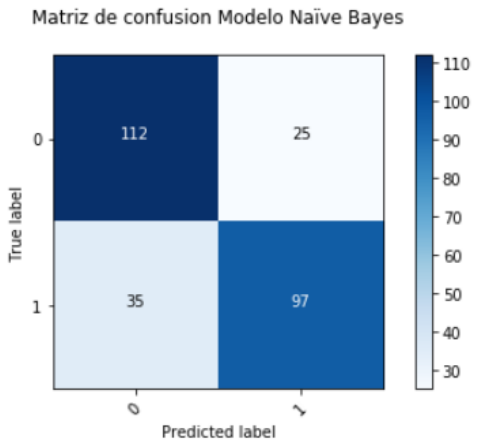

Figura 4.3. Matriz de confusión del Clasificador Naïve Bayes para el conjunto de datos de prueba.

Para el conjunto de prueba, de acuerdo con la matriz de confusión, el modelo clasifica correctamente 112 mujeres como no diabéticas y a 97 mujeres como diabéticas que, en realidad, lo son.

Por el contrario, clasifica de forma incorrecta a 25 mujeres no diabéticas como diabéticas y a 35 mujeres diabéticas como no diabéticas. De esta forma, el modelo tiene una tasa de acierto del 77.69%. Este porcentaje resulta de dividir (112+97) clasificadas correctamente entre 269 mujeres que componen el conjunto de prueba. Esta tasa de acierto coincide con el *accuracy*, debido a que, por definición, es la misma métrica.

4.3.3.2 Resultados del Clasificador Random Forest

El Clasificador Random Forest es más sofisticado que el Clasificador Naïve Bayes. Se basa en trabajar con una colección de árboles de clasificación y promediar sus resultados. Por este motivo, se espera que el ajuste sea algo superior al Clasificador Naïve Bayes, basado tan sólo en el cálculo de probabilidades.

prediction	Outcome
0.0	0.0
0.0	0.0
0.0	0.0
0.0	0.0
0.0	0.0
0.0	0.0
0.0	0.0
0.0	0.0
0.0	0.0
0.0	0.0
0.0	0.0
1.0	0.0
0.0	0.0
1.0	1.0
0.0	0.0
0.0	0.0
1.0	1.0
0.0	0.0
1.0	1.0
0.0	0.0

only showing top 20 rows

Tabla 4.14. Valores de la predicción del Clasificador Random Forest para el conjunto de datos de prueba.

Lo señalado en el párrafo anterior se corrobora observando los valores de la predicción realizada por el modelo para el conjunto de test reflejados en la Tabla 4.14. Para esta muestra de 20 mujeres ilustrada en la tabla señalada, clasifica correctamente a la mayoría de ellas. La predicción coincide con la realidad para 2 de las 3 mujeres diabéticas y 17 de las 18 mujeres no diabéticas. Es decir, tan sólo clasifica de forma incorrecta a una de las 20 mujeres.

```
Bondad del ajuste del Clasificador Random Forest
.............................................

 Error conjunto de test = 0.112971
 Accuracy conjunto de test = 0.887029
_____________________________________________

 Error conjunto de entrenamiento = 0
 Accuracy conjunto de entrenamiento= 1
.............................................
```

Tabla 4.15. Bondad del ajuste del Clasificador Random Forest.

Esto se corrobora mediante la métrica *accuracy* de bondad del ajuste. Para el conjunto de entrenamiento, el modelo alcanza una precisión del 100%. En el caso del conjunto de prueba, esta precisión disminuye al 88.70% y, por ende, tan sólo falla para el 11.29% de las mujeres, un resultado que sigue siendo muy bueno para datos reales.

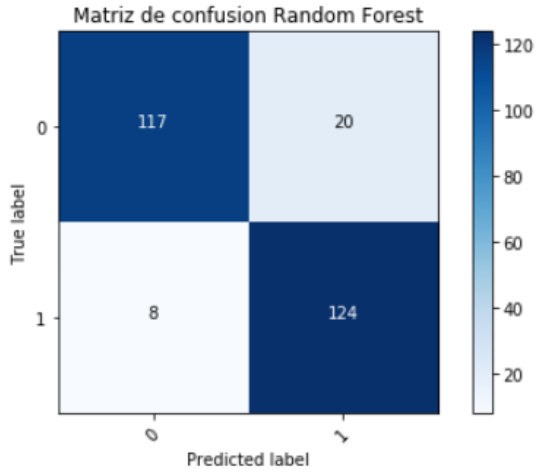

Figura 4.4. Matriz de confusión del Clasificador Random Forest para el conjunto de datos de prueba.

Estos resultados se pueden ver de forma detallada en la matriz de confusión generada para el conjunto de prueba (Figura 4.4.). El Clasificador Random Forest es capaz de clasificar correctamente a 117 mujeres no diabéticas como no diabéticas y a 124 mujeres diabéticas como diabéticas.

La tasa de error del 10.41% en la predicción viene de que el modelo clasifica incorrectamente a 8 mujeres diabéticas como diabéticas y a 20 mujeres no diabéticas como diabéticas. Es decir, falla en 30 mujeres de las 269 que componen el conjunto de prueba. Por lo que se puede concluir que el modelo tiene una gran capacidad de generalización ante instancias desconocidas.

4.3.3.3 Resultados del algoritmo Linear Support Vector Classification

Para terminar con la fase de interpretación de resultados, se comentan los resultados obtenidos por la Máquina de Vector Soporte Lineal para clasificación o Linear Support Vector Classification.

Según la literatura, la Máquina de Vector Soporte es el modelo que, en general, proporciona el mejor ajuste en clasificación. Esto se debe a que basa su funcionamiento en construir una función o hiperplano óptimo que optimice la predicción de las instancias en base a sus clases. Para ello, se vale también de otras dos funciones que le hacen de soporte o vectores de soporte. Estos se emplean para diferenciar a un lado y a otro del hiperplano las instancias de cada clase.

Outcome	Prediction
0.0	0.0
0.0	0.0
0.0	0.0
0.0	1.0
0.0	0.0
0.0	0.0
0.0	0.0
0.0	0.0
0.0	0.0
0.0	1.0
0.0	0.0
0.0	1.0
0.0	0.0
1.0	1.0
0.0	0.0
0.0	0.0
1.0	1.0
0.0	1.0
1.0	1.0
0.0	1.0

only showing top 20 rows

Tabla 4.16. Valores de la predicción del algoritmo Linear Support Vector Classification para el conjunto de datos de prueba.

El algoritmo LSVC, como se puede observar en la Tabla 4.16. para una muestra de 20 mujeres del total de mujeres conjunto de prueba, es capaz de clasificar correctamente como diabéticas a todas las mujeres que lo son según la variable *"Outcome"*. Por el contrario, clasifica como diabéticas a un gran número de mujeres que no lo son.

VALORES DE LOS COEFICIENTES E INTERCEPTO MODELO LSVC

Pregnancies	0.09121522263295011
Glucose	0.015832504727717026
BloodPressure	-0.05696561272847348
SkinThickness	0.013521382497163403
Insulin	0.008098565663054006
BMI	0.002719465300481087
DiabetesPedigreeFunction	0.4214429964197816
Age	0.007769733307770825
Intercepto	-0.0832372646476921

Tabla 4.17. Coeficientes e intercepto del algoritmo Linear Support Vector Classification.

La tabla 4.17. muestra los coeficientes arrojados para las distintas variables explicativas del modelo y el valor del intercepto o constante.

En este caso, nos limitamos a exponer los resultados y no nos detendremos a detallar qué interpretación se le puede dar a cada coeficiente y al intercepto que genera como resultado el algoritmo. Comprender perfectamente todos los resultados implicaría ser expertos en este algoritmo (al igual que en los anteriores) y conocer, entre otros muchos aspectos, la configuración de los parámetros. Además, el objetivo de ilustrar con un ejemplo práctico el proceso de KDD aplicando distintos algoritmos es completar de forma práctica la descripción teórica que se ha hecho de él.

```
_____________________________________________________

Bondad de ajuste del Clasificador LSVC
.....................................................
Accuracy conjunto de entrenamiento  = 0.7373461012311902

Accuracy conjunto de test  = 0.7620817843866171
_____________________________________________________
```

Tabla 4.18. Bondad del ajuste del algoritmo Linear Support Vector Classification.

Si se evalúa la bondad de ajuste, el modelo es capaz de clasificar adecuadamente el 73.73% de las mujeres pertenecientes al conjunto de entrenamiento. Este valor aumenta hasta el 76.21% para las mujeres pertenecientes al conjunto de prueba. Por lo que se puede concluir, que el modelo generaliza correctamente.

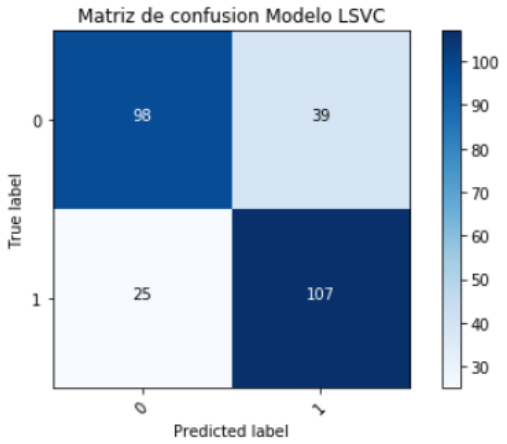

Figura 4.5. Matriz de confusión del Clasificador LSVC para el conjunto de datos de prueba.

De este modo, la Máquina de Vector Soporte Lineal clasifica correctamente a 107 mujeres como diabéticas y a 98 mujeres como no diabéticas, que realmente lo son. Por el contrario, clasifica

incorrectamente, a 25 mujeres diabéticas como no diabéticas y a 39 mujeres no diabéticas como diabéticas.

Para terminar con este apartado que recoge la fase de interpretación del proceso de KDD, se muestra una tabla resumen con la bondad del ajuste de los distintos modelos.

```
_________________________________________________________________

          Bondad de ajuste de los distintos clasificadores
.................................................................

Accuracy Naïve Bayes conjunto de entrenamiento  =  77.00394218134035
Accuracy LSVC conjunto de entrenamiento  =  75.42706964520369
Accuracy Random Forest conjunto de entrenamiento  =  100.0

.................................................................

Accuracy Naïve Bayes conjunto de test  =  77.40585774058577
Accuracy LSVC conjunto de test  =  77.82426778242679
Accuracy Random Forest conjunto de test  =  88.70292887029288
_________________________________________________________________
```

Tabla 4.19. Bondad del ajuste de los clasificadores empleados.

Como se puede observar en la Tabla 4.19., el modelo Random Forest clasifica correctamente todas las instancias para el conjunto de entrenamiento y también proporciona un ajuste muy bueno para el conjunto de test. Por lo que se convierte en modelo de clasificación con mayor capacidad de predicción para este conjunto de datos de ejemplo.

5 Conclusiones

El proceso del KDD, como es conocido, tiene cierta complejidad intrínseca que se supera con la experiencia y, para ser desarrollado en entornos de Big Data, además, requiere subir un nivel más de complejidad pues el uso de los recursos de programación para Big Data, no es trivial.

No se debe olvidar que es un proceso que va de la mano con el Análisis Exploratorio de Datos en la mayoría de ocasiones, pues conocer los datos es fundamental, dado que el KDD es un proceso que no es completamente sistemático, sino que se aplica en función de lo que ese va obteniendo, es decir, esta muy vinculado al conjunto de datos empleado. Por tanto, este último, sin duda, ayuda a sacar aún más partido a los datos y a comprender mejor el conocimiento oculto en los datos en crudo.

En relación a las tecnologías necesarias, es indudable el reconocimiento que posee Apache Spark, en gran medida alcanzado por su capacidad para tratar incluso datos en tiempo real, muy necesaria en entornos Big Data, superando así al procesamiento tradicional en batches realizado por Apache Hadoop. A esto hay que añadir, como se ha podido ver en el ejemplo, la significativa mayor facilidad de uso de la tecnología que proporciona Apache Spark, en concreto, en su API de Python, que pone a disposición del científico de datos una amplia gama de librerías que le permiten realizar todo el trabajo que conllevan estos procesos de forma fácil, rápida y sencilla, pese a que pueda pensarse lo contrario por tratarse de datos con unas características muy específicas y, sobre todo, con una gran complejidad.

Aunque no hay que olvidar que se trata de dos tecnologías que se complementan. Apache Spark trabaja mucho más rápido, entre otras virtudes, pero necesita del sistema de almacenamiento de ficheros HDFS de Apache Hadoop para poder trabajar correctamente en entornos Big Data, por tanto, pese a que se pueda pensar lo contrario, ambas tecnologías van perfectamente de la mano.

Hay que destacar por último que se trata de un proceso y un análisis asociados a una tecnología que evoluciona a gran velocidad. Por ello, siempre queda la puerta abierta a la mejora continua, no sólo de estas tecnologías, sino que también del propio proceso y análisis. Posiblemente, en unos años, se sumen nuevas técnicas de preprocesamiento y nuevos algoritmos a los ya existentes para obtener un mejor conocimiento a partir de los datos, así como nuevos tipos de análisis y formas de relaciones que descubrir.

Referencias

Ahsaan, S. U., & Mourya, A. K. (2019). Big data analytics: challenges and technologies. *Annals of the Faculty of Engineering Hunedoara*, 17(4), 75-79.

Apache. (2019). *Welcome to Apache Hadoop!* Recuperado de https://hadoop.apache.org/old/index.pdf

Apache Spark. (s. f.-a). *Classification and regression - Spark 3.0.0 Documentation - Linear Support Vector Machine*. Apache Spark 3.0.0. Recuperado 6 de julio de 2020, de https://spark.apache.org/docs/latest/ml-classification-regression.html#linear-support-vector-machine

Apache Spark. (s. f.-b). *Classification and regression - Spark 3.0.0 Documentation - Naive Bayes.* Apache Spark 3.0.0. Recuperado 6 de julio de 2020, de https://spark.apache.org/docs/latest/ml-classification-regression.html#naive-bayes

Apache Spark. (s. f.-c). *Classification and regression - Spark 3.0.0 Documentation - Random Forest*. Apache Spark 3.0.0. Recuperado 6 de julio de 2020, de https://spark.apache.org/docs/latest/ml-classification-regression.html#random-forest-classifier

Apache Spark 2.2.0. (s. f.). *Extracting, transforming and selecting features* - Spark 2.2.0 Documentation. Apache Spark 2.2.0. Recuperado 8 de julio de 2020, de https://spark.apache.org/docs/2.2.0/ml-features.html#chisqselector

Atria Innovation. (22 de octubre de 2019). [Diagrama de una red neuronal]. Recuperado de https://www.atriainnovation.com/que-son-las-redes-neuronales-y-sus-funciones/

Beakta, R. (2015). *Big Data And Hadoop: A Review Paper*. International Journal of Advanced Research in Computer Science y Technology (IJARCST 2014), 2(2), 214-2017. Recuperado de https://www.researchgate.net/publication/281403776_Big_Data_And_Hadoop_A_Review_Paper

Becher, J. D., Berkhin, P., & Freeman, E. (2000). Automating exploratory data analysis for efficient data mining. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD* '00, 424-429. https://doi.org/10.1145/347090.347179

Camacho, J., Pérez, A., Rodríguez, R., & Jiménez-Mañas, E. (2015). josecamachop/MEDA-Toolbox. *Chemometrics and Intelligent Laboratory Systems*, 143: 49-57. Recuperado 5 de julio de 2020, de https://github.com/jose camachop/MEDA-Toolbox

Cano, A., Ventura, S., & Cios, K. J. (2014). Scalable CAIM discretization on multiple GPUs using concurrent kernels. *The Journal of Supercomputing*, 69(1), 273-292. https://doi.org/10.1007/s11227-014-1151-8

Cerquides, J., & de Mántaras, R. L. (1997). Proposal and empirical comparison of a parallelizable distance-based discretization method. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, KDD'97*, 139–142.

Che, D., Safran, M., & Peng, Z. (2013). From Big Data to Big Data Mining: Challenges, Issues, and Opportunities. *Database Systems for Advanced Applications*, 1-15. https://doi.org/10.1007/978-3-642-40270-8_1

Chon Ho, Y. (2010). Exploratory data analysis in the context of data mining and resampling. *International Journal of Psychological Research*, 3(1), 9-22. https://doi.org/10.21500/20112084.819

Dean, J., & Ghemawat, S. (2008). *MapReduce: simplified data processing on large clusters*. Communications of the ACM, 51(1), 107-113.

Doulkeridis, C., & Nørvåg, K. (2013). A survey of large-scale analytical query processing in MapReduce. *The VLDB Journal*, 23(3), 355-380. https://doi.org/10.1007/s00778-013-0319-9

García-Gil, D., Luengo, J., García, S., & Herrera, F. (2019). Enabling smart data: Noise filtering in big data classification. *Information Sciences*, 479, 135–152

García, S., Luengo, J., Sáez, J. A., López, V., & Herrera, F. (2013). A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(4), 734–750.

García, S., Luengo, J., & Herrera, F. (2015). *Data preprocessing in data mining*. New York: Springer.

García, S., Ramírez-Gallego, S., Luengo, J., & Herrera, F. (2016). Big Data: Preprocesamiento y calidad de datos. *novática*, 237, 17-23. Recuperado de http://150.214.190.154/sites/default/files/ficherosPublicaciones/2133_Nv237-Digital-sramirez.pdf

Fases del Proceso de Knowledge Discovery in Databases. (s. f.). [Ilustración]. Recuperado de https://aulasvirtuales.uhu.es/pluginfile.php/284794/mod_resource/content/6/Tema%202_2020_V1-p1-57.pdf

Hall, M. A. (1999). *Correlation-based feature selection for Machine Learning*. Hamilton. Department of Computer Science, Waikato University.

Hariri, R. H., Fredericks, E. M., & Bowers, K. M. (2019). Uncertainty in big data analytics: survey, opportunities, and challenges. *Journal of Big Data*, 6(1), 44. https://doi.org/10.118 6/s40537-019-0206-3

Hernández Yeja, A. (julio, 2015). *Flujo Lógico de procesos MapReduce* [Figura]. Recuperado de https://www.researchgate.net/figure/Figura-1-Flujo-Logico-de-procesos-MapReduce_fig1_292137101

History | Apache Spark. (s. f.). Recuperado 16 de junio de 2020, de https://spark.apache.org/history.html

Jolliffe, I. (2011). *Principal Component Analysis*. Berlin, Alemania: Springer

Kaggle. (6 octubre, 2016). Pima Indians Diabetes Database [Conjunto de datos]. National Institute of Diabetes and Digestive and Kidney Diseases. https://www.kaggle.com/uciml/pim a-indians-diabetes-database

Lämmel, R. (2008). *Google's MapReduce programming model - Revisited.* Science of Computer Programming, 70(1), 1-30. https://doi.org/10.1016/j.scico.2007.07.001

Lin, J.J. (2012). MapReduce is Good Enough? If All You Have is a Hammer, Throw Away Everything That's Not a Nail! *Big data*, 1 1, 28-37.

Liu, H., & Motoda, H. (2002). On issues of instance selection. *Data Mining and Knowledge Discovery*, 6(2), 115–130 (2002)

Luengo, J., García-Gil, S., Ramírez-Gallego, S., & Herrera, F. (2020). *Big Data Preprocessing*. New York, Estados Unidos: Springer Publishing. https://doi.org/10.1007/978-3-030-39105-8

Lugat, V. (27 enero, 2020). Pima Indians Diabetes - EDA & Prediction (0.906). kaggle. Recuperado 5 de julio de 2020, de https://www.kaggle.com/vincentlugat/pima-indians-diabetes-eda-prediction-0-906/data

Madaka, Y. (28 junio, 2019). *Building an ML application using MLlib in Pyspark*. towards data science. https://towardsdatascience.com/building-an-ml-application-with-mllib-in-pyspark-part-1-ac13f01606e2

Maimon, O., & Rokach, L. (2010). *Data Mining and Knowledge Discovery Handbook* (2.a ed.). New York, Estados Unidos: Springer Publishing. https://doi.org/10.1007/978-0-387- 09823-4

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... & Xin, D. (2016). Mllib: Machine learning in Apache Spark. *The Journal of Machine Learning Research*, *17*(1), 1235-1241.

NIST Sematech. (2006). What is EDA? Recuperado de https://www.itl.nist.gov/div898/handbook/eda/section1/eda11.htm

Opensource.com. (2014, 26 agosto). *Apache Hadoop Ecosystem* [Imagen]. Recuperado de https://opensource.com/sites/default/files/resize/styles/image-full-size/public/images/life-uploads/hadoop-EcoSys_yarn-640x418.PNG

Organización Mundial de la Salud. (s. f.). *OMS | 10 datos sobre la obesidad*. Organización Mundial de la Salud. Recuperado 7 de julio de 2020, de https://www.who.int/features/factfiles/obesity/facts/es/

Pérez Marqués, M. (2015). *Big Data. Técnicas, herramientas y aplicaciones*. Recuperado de http://rclibros.es/wp-content/uploads/2015/04/capitulo_9788494305559.pdf

Pyle, D. (1999). *Data preparation for data mining.* San Francisco, Estados Unidos: Morgan Kaufmann.

Rathi, D. R., & Lohiya, S. (2014). Big data and hadoop. *International Journal of Advanced Research in Computer Science y Technology* (IJARCST 2014), 2(2).

Riahi, Y., & Riahi, S. (2018). Big Data and Big Data Analytics: concepts, types and technologies. *International Journal of Research and Engineering*, 5(9), 524-528. https://doi.org/10.21276/ijre.2018.5.9.5

Riquelme Santos, J. C., Ruiz, R. & Gilbert, K. (2006). *Minería de datos: Conceptos y tendencias. Inteligencia Artificial: Revista Iberoamericana de Inteligencia Artificial, 10 (29), 11-18.*

Salvador Figueras, M & Gargallo, P. (2003). *Análisis Exploratorio de Datos.* Recuperado 20 de junio de 2020, de http://www.5campus.com/leccion/aed

Schenkel, R. (s.f.). *MapReduce*. Max Planck Institute Informatik By MCI-Cluster of Excellence. Recuperado 30 de junio de 2020, de http://resources.mpi-inf.mpg.de/d5/teaching/ss10/dyn/slides/MapReduce.pdf

Scikit-learn. (s. f.). Confusion matrix — scikit-learn 0.18.2 documentation. Recuperado 6 de julio de 2020, de https://scikit-learn.org/0.18/auto_examples/model_selection/plot_confusion_matrix.html

Sitio big data. (24 diciembre, 2019). Ecosistema Apache Spark [Figura]. sitio big data. https://sitiobigdata.com/wp-content/uploads/2019/12/apache-spark-intro-fig5.jpg

Smith, J. W., Everhart, J. E., Dickson, W. C., Knowler, W. C., & Johannes, R. S. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, 261-265. IEEE Computer Society Press. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2245318/pdf/procascamc00018-0276.pdf

Spark ML Programming Guide - Spark 1.2.2 Documentation. (s. f.). Recuperado 16 de junio de 2020, de https://spark.apache.org/docs/1.2.2/ml-guide.html

Spark SQL and DataFrames - Spark 2.3.0 Documentation. (s. f.). Recuperado 16 de junio de 2020, de https://spark.apache.org/docs/2.3.0/sql-programming-guide.html#datasets-and-dataf rames

Staniak, M., & Biecek, P. (2019). The Landscape of R Packages for Automated Exploratory Data Analysis. *The R Journal*, 11(2), 347. https://doi.org/10.32614/rj-2019-033

StataCorp LLC. (s. f.). Stata | Why Stata. Recuperado 4 de julio de 2020, de https://www.stata.com/why-use-stata/

Suguna, S. (2016). Improvement of HADOOP Ecosystem and Their Pros and Cons in Big Data. *International Journal Of Engineering And Computer Science*, 1-7. https://doi.org/10.18535/ijecs/v5i5.57

Tarrés, M. C., Moscoloni, N., Navone, H., & D'Ottavio, A. E. (2016). ANÁLISIS MULTIDIMENSIONAL DE UNA BASE DE DATOS DE MUJERES PIMA/ MULTIDIMENSIONAL ANALYSIS FROM A DATABASE OF PIMA WOMEN. *Biotecnia*, 18(3), 14-19. https://doi.org/10.18633/biotecnia.v18i3.330

Thakur, V. K., Harshit, V., Utkarsh, R., Parmod, J., & Amit, C. (2020). Review Paper on Big Data Analytics. *International Journal for Research in Applied Science and Engineering Technology*, 8(6), 785-788. https://doi.org/10.22214/ijraset.2020.6126

Triguero, I., Peralta, D., Bacardit, J., García, S., & Herrera, F. (2015). MRPR: A MapReduce solution for prototype reduction in big data classification. *Neurocomputing*, 150, Part A, 331–345, 2015.

Tsai, C., Lai, C., Chao, H. et al. *Big data analytics: a survey.* Journal of Big Data 2, 21 (2015). https://doi.org/10.1186/s40537-015-0030-3

Tukey, J. W. (1977). *Exploratory Data Analysis.* Van Haren Publishing.

user8183279. (19 junio, 2017). *How to find count of Null and Nan values for each column in a Pyspark dataframe efficiently?* [Publicación en foro online]. Mensaje publicado en https://stackoverrun.com/es/q/12259557

Vaidya, G. M., & Kshirsagar, M. M. (2020). A Survey of Algorithms, Technologies and Issues in Big Data Analytics and Applications. *2020 4th International Conference on Intelligent Computing and Control Systems* (ICICCS), 347-350. Madurai, India: IEEE. https://doi.org/10.1109/iciccs48265.2020.9121064

Veith, A., & Assunçao, M. (2019). Apache Spark. En *Encyclopedia of Big Data Technologies* (pp. 1-5). Nueva York, Estados Unidos: Springer Publishing. https://doi.org/10.1007/978-3-319-63962-8_37-1

Wani, M. A., & Jabin, S. (2017). Big Data: Issues, Challenges, and Techniques in Business Intelligence. *Advances in Intelligent Systems and Computing*, 613-628. https://doi.org/10.1007/978-981-10-6620-7_59

Withanawasam, J. (2015). *Apache Mahout Essentials. Zaltbommel*, Países Bajos: Van Haren Publishing. Recuperado de http://2.droppdf.com/files/EIY3V/apache-mahout-essentials .pdf

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., ... & Stoica, I. (2012). *Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing.* In Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12) (pp. 15-28).

Zhao, Y., Niu, Z., Peng, X., & Dai, L. (2011). A Discretization Algorithm of Numerical Attributes for Digital Library Evaluation Based on Data Mining Technology. *Digital Libraries: For Cultural Heritage, Knowledge Dissemination, and Future Creation*, 70-76. https://doi.org/10.1007/978-3-642-24826-9_12

Apéndices

Librerías importadas

In [290]:

```
#Iniciamos el entorno Spark para poder ser utilizado en Jupyter notebook
#Cargamos las librerias que vamos a necesitar
import findspark
findspark.init()
findspark.find()
import pyspark
findspark.find()

from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.classification import LinearSVC
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.mllib.stat import Statistics
from pyspark.ml.stat import Correlation
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import MinMaxScaler
from pyspark.ml.feature import ChiSqSelector
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTEENN
from collections import Counter

import seaborn as sns
import sklearn
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt
import plotly.graph_objs as go
import plotly.offline as py
import plotly.tools as tls
import plotly.figure_factory as ff
```

Código para la imputación de valores perdidos por la media de cada variable

In [17]:

```
# Definimos una función que calculará la media en base al valor de salida al que pertenece para imputar luego los valores
    # Referencias: https://www.kaggle.com/vincentlugat/pima-indians-diabetes-eda-prediction-0-906/notebook

dfpandas=df.toPandas()
D = dfpandas[(df_pandas['Outcome'] != 0)]
H = dfpandas[(df_pandas['Outcome'] == 0)]


def median_target(var):
    temp = dfpandas[dfpandas[var].notnull()]
    temp = temp[[var, 'Outcome']].groupby(['Outcome'])[[var]].median().reset_index()
    return temp

def plot_distribution(data_select, size_bin) :
    # 2 datasets
    tmp1 = D[data_select]
    tmp2 = H[data_select]
    hist_data = [tmp1, tmp2]

    group_labels = ['diabetic', 'healthy']
    colors = ['#FFD700', '#7EC0EE']

    fig = ff.create_distplot(hist_data, group_labels, colors = colors, show_hist = True,
                             bin_size = size_bin, curve_type='kde')

    fig['layout'].update(title = data_select)

    py.iplot(fig, filename = 'Density plot')
```

Código para la representación de las distintas funciones de distribución de las variables, impresión por pantalla de los valores de la media para cada clase e imputación de valores perdidos

In [16]:

```
# Función de distribución de la insulina
print('Función de Distribución de la Insulina')
plot_distribution('Insulin', 0)

# Media de la insulina según si el paciente es o no diabético
print('Media de la insulina según si el paciente es o no diabético\n')
print(median_target('Insulin'))

# Reemplazamos los valores perdidos por la media para diabéticos y no diabéticos
df_pandas.loc[(df_pandas['Outcome'] == 0 ) & (df_pandas['Insulin'].isnull()), 'Insulin'] = 102.5
df_pandas.loc[(df_pandas['Outcome'] == 1 ) & (df_pandas['Insulin'].isnull()), 'Insulin'] = 169.5

print('\n------------------------------\n')

# Función de distribución de la glucosa
print('Función de Distribución de la glucosa')
plot_distribution('Glucose', 0)

# Media de la glucosa según si el paciente es o no diabético
print('Media de la glucosa según si el paciente es o no diabético\n')
print(median_target('Glucose'))

# Reemplazamos los valores perdidos por la media para diabéticos y no diabéticos
df_pandas.loc[(df_pandas['Outcome'] == 0 ) & (df_pandas['Glucose'].isnull()), 'Glucose'] = 107
df_pandas.loc[(df_pandas['Outcome'] == 1 ) & (df_pandas['Glucose'].isnull()), 'Glucose'] = 140

print('\n------------------------------\n')
```

```
# Función de distribución de la tensión arterial
print('Función de Distribución de la tensión arterial')
plot_distribution('BloodPressure', 0)

# Media de la tensión arterial según si el paciente es o no diabético
print('Media de la tensión arterial según si el paciente es o no diabético\n')
print(median_target('BloodPressure'))

# Reemplazamos los valores perdidos por la media para diabéticos y no diabéticos
df_pandas.loc[(df_pandas['Outcome'] == 0 ) & (df_pandas['BloodPressure'].isnull()), 'BloodPressure'] = 70
df_pandas.loc[(df_pandas['Outcome'] == 1 ) & (df_pandas['BloodPressure'].isnull()), 'BloodPressure'] = 74

print('\n------------------------------\n')

# Función de distribución del grosor de la piel
print('Función de Distribución del grosor de la piel')
plot_distribution('SkinThickness', 0)

# Media del grosor de la piel según si el paciente es o no diabético
print('Media del Índice de Masa Corporal según si el paciente es o no diabético\n')
print(median_target('SkinThickness'))

# Reemplazamos los valores perdidos por la media para diabéticos y no diabéticos
df_pandas.loc[(df_pandas['Outcome'] == 0 ) & (df_pandas['SkinThickness'].isnull()), 'SkinThickness'] = 21
df_pandas.loc[(df_pandas['Outcome'] == 1 ) & (df_pandas['SkinThickness'].isnull()), 'SkinThickness'] = 27

print('\n------------------------------\n')
```

```
# Función de distribución del Índice de Masa Corporal
print('Función de Distribución del Índice de Masa Corporal')
plot_distribution('BMI', 0)

# Media del Índice de Masa Corporal según si el paciente es o no diabético
print('Media del Índice de Masa Corporal según si el paciente es o no diabético\n')
print(median_target('BMI'))

# Reemplazamos los valores perdidos por la media para diabéticos y no diabéticos
df_pandas.loc[(df_pandas['Outcome'] == 0 ) & (df_pandas['BMI'].isnull()), 'BMI'] = 30.05
df_pandas.loc[(df_pandas['Outcome'] == 1 ) & (df_pandas['BMI'].isnull()), 'BMI'] = 34.24
```

Código para la comprobación de la no existencia de valores perdidos en el conjunto de datos.

In [9]:
```
# Comprobamos que no quedan valores perdidos

missing_plot(df_pandas, 'Outcome')

print('ESTADÍSTICOS BÁSICOS DEL DATASET CORREGIDO')
df_pandas.describe()
```

Código que guarda el conjunto de datos tras el EDA y lo carga para iniciar el proceso de KDD.

```
# Guardamos el Dataset para poder seguir trabajando con él en las siguientes fases
df_pandas.to_csv('diabetesprep.csv')

#Cargamos y mostramos el nuevo dataset con el que trabajaremos a partir de ahora
dfprep = spark.read.csv('diabetesprep.csv', header=True, inferSchema=True)
dfprep.show(5)
```

Código empleado para la corrección del desbalanceo entre clases

In [81]:
```
# A continuación, corregiremos el desbalanceo entre clases a partir del dataset nuevo

print('\nBalance de clases del dataset:')#Mostramos el balanceo de las clases
enfermo = dfprep.select("Outcome").filter((col("Outcome") == 1)).count()
no_enfermo = dfprep.select("Outcome").filter((col("Outcome") == 0)).count()
print("Numero de datos de no_enfermos -0-: "+str(no_enfermo)+" ("+str((no_enfermo/(no_enfermo+enfermo))*100)+"%)")
print("Numero de datos de enfermos -1-: "+str(enfermo)+" ("+str((enfermo/(no_enfermo+enfermo))*100)+"%)")


print('\nBalance de clases del dataset - NUEVO BALANCE DE CLASES:')

variables=dfprep.toPandas().filter(items=["Pregnancies", "Glucose", "BloodPressure", "SkinThickness",
                                          "Insulin", "BMI", "DiabetesPedigreeFunction", "Age"])
vsalida = dfprep.toPandas().filter(items=['Outcome'])

from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42, sampling_strategy='auto')
variables_res, vsalida_res = smote.fit_sample(variables, vsalida)
#print('Forma del nuevo dataset balanceado {}'.format(Counter(vsalida_res)))

dataframe_1 = pd.DataFrame(variables_res,columns=["Pregnancies", "Glucose", "BloodPressure", "SkinThickness",
                                                  "Insulin", "BMI", "DiabetesPedigreeFunction", "Age"])
dataframe_2 = pd.DataFrame(vsalida_res, columns = ['Outcome'])

datos_pd = dataframe_1.combine_first(dataframe_2)
datos = spark.createDataFrame(datos_pd)

#Mostramos el balanceo de las clases

enfermo = datos.select("Outcome").filter((col("Outcome") == 1)).count()
no_enfermo = datos.select("Outcome").filter((col("Outcome") == 0)).count()
print("Numero de datos de no diabéticos (0): "+str(no_enfermo)+" ("+str((no_enfermo/(no_enfermo+enfermo))*100)+"%)")
print("Numero de datos de diabéticos (1): "+str(enfermo)+" ("+str((enfermo/(no_enfermo+enfermo))*100)+"%)")
```

Código empleado para la normalización de las variables del conjunto de datos

In [55]:
```
# Normalizamos el dataset

#Normalizamos el dataset
print("\nDATASET NORMALIZADO:")

# Generamos una columna nueva 'features' que contendrán todas las columnas de caracteristicas del dataframe
df_indexado = VectorAssembler(inputCols=["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI",
                                         "DiabetesPedigreeFunction", "Age"], outputCol="features").transform(datos)

# Transformamos el dataset a MinMaxScaler
scaler = MinMaxScaler(inputCol="features", outputCol="features_Scaled")

# Pipeline con VectorAssembler y MinMaxScaler
pipeline_Norm = Pipeline(stages=[scaler])

# Ajustamos la pipeline al dataframe
df_Norm = pipeline_Norm.fit(df_indexado).transform(df_indexado)

print(df_Norm.select("features").show(5, False)) #False para no truncar los datos y mostrar todas los valores de la columna
print(df_Norm.select("features_Scaled").show(5, False))

print("\n----------------------------------------\n")
```

Código empleado para la selección de características del conjunto de datos

In [25]:
```
# Selección de características

# Realizamos una seleccion de caracteristicas basado en la entropía entre ellas, utilizando la funcion ChiSqSelector
# Se generará una nueva columna 'selectedFeatures' que contendrá los valores de las columnas que haya elegido la función
# Podemos elegir la columna de caracteristicas normalizadas (features_Scaled) o sin normalizar (features)

selector = ChiSqSelector(numTopFeatures=8, featuresCol="features", outputCol="selectedFeatures", labelCol="Outcome")
result = selector.fit(df_Norm).transform(df_Norm)
print("ChiSqSelector output with ---[ "+str(result.head().selectedFeatures)+" ]--- features selected")

```

Código empleado para la representación de la matriz de confusión generada por la predicción arrojada por los distintos clasificadores

In [94]:

```
# Definiremos una función que represente la matriz de confusión para los distintos modelos
# De esta forma, tendremos otra métrica más que ayude a comparar la bondad del ajuste de los distintos modelos

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    Esta funcion muestra la matriz de confusion de forma gráfica.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Matriz de confusion normalizada")
    else:
        print('Matriz de confusion')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```