UNIVERSIDAD DE GRANADA

Trabajo Fin de Máster

Tratamiento de falta de información en técnicas de minería de datos

Laura Freire Míguez

2022/2023

UNIVERSIDAD DE GRANADA

Máster Estadística Aplicada

Trabajo Fin de Máster

Tratamiento de falta de información en técnicas de minería de datos

Laura Freire Míguez
Tutora: Rocío Raya Miranda

Febrero, 2023

UNIVERSIDAD DE GRANADA

Declaración de autoría y originalidad del Trabajo de Fin de Máster

Considerando que la presentación de un trabajo hecho por otra persona o la copia de textos, fotos y gráficas sin citar su procedencia se considera plagio, yo Laura Freire Míguez con DNI 53820787M, que presenta el Trabajo Fin de Máster con el título *Tratamiento de falta de información en técnicas de minería de datos* declara la autoría y asume la originalidad de este trabajo, donde se han utilizado distintas fuentes que han sido todas citadas debidamente en la memoria.

Y para que así conste firmo el presente documento en Granada a 21 de enero de 2023.

Laura Freire Míguez

Resumen

Los datos faltantes son muy comunes a la hora de tratar con datos reales y estos repercuten directamente en el análisis, afectando a su calidad y exactitud de los resultados. Pueden ser causados por una variedad de factores, como errores de recolección de datos, problemas de calidad de datos, omisión de observaciones u otros motivos.

A la hora de implementar una técnica de minería de datos, dependiendo del tipo de datos faltantes ante los que nos encontremos, el enfoque para lidiar con ellos puede variar. Muchas veces resulta conveniente aplicar un método de imputación para aproximar estos valores y poder realizar el análisis de la forma más completa posible. Existen numerosos métodos de imputación, la elección de cual utilizar depende tanto de los datos como del tipo de técnica que se va a realizar con ellos. Estos métodos pueden ser simples o complejos, dependiendo de la cantidad y complejidad de los datos faltantes. Los métodos simples incluyen la imputación por la media, la mediana y la moda. Los métodos más complejos incluyen la imputación basada en modelos de regresión y la imputación basada en los valores más próximos, entre otras.

En el primer capítulo de este trabajo vamos a introducir el concepto de minería de datos, su origen, su relación con el proceso *KDD* y las distintas técnicas que engloba este campo con la correspondiente clasificación de estas en métodos supervisados y no supervisados. Además se mecionarán las aplicaciones más frecuentes de estas técnicas, en particular, y de la minería de datos, en general.

A continuación se clasifican los datos faltantes según su mecanismo de inducción, distinguiendo entre la falta completamente al azar ($MCAR$), la falta al azar (MAR) y no faltante al azar ($NMAR$). Se ven las alternativas a realizar ante la presencia de estos datos, desde la eliminación o exclusión, a la imputación, y los principales métodos de imputación

para aproximarlos. Como se ha mencionado, entre estos métodos se encuentran algunos más simples como la imputación mediante la media, la mediana o la moda y otros más complejos, que incluyen la estimación e imputación por valores cercanos, como la imputación mediante *KNN*, *K-means*, *EC*, *EM*, *PMM*, etc.

Se hace hincapié en la importancia de validar esta imputación y hacer un análisis de sensibilidad sobre los resultados para valorar la idoneidad del método, para ello se mencionan métricas como la sensibilidad, exactitud, F1 *score*, curva *ROC*, *Silhouette*, ...

Se trata también el impacto de los datos faltantes en técnicas de minería de datos concretas como en los algoritmos *KNN*, árboles de decisión, normas de asociación o redes neuronales.

Finalmente, en el último capítulo, se realiza una aplicación práctica de lo tratado hasta ese momento con datos reales mediante el software estadístico R. En esta apliccación se incluye una descripción de los datos utilizados, que recogen distintas variables sobre el entorno de un niño para evaluar y recomendar o no la admisión de este en la guardería infantil, en el primer caso, y distintas características de las setas para concluír si son comestibles o venenosas, en el segundo. A estos datos se le aplican cuatro técnicas de imputación de datos faltantes distintas (así como se incluye la comparación entre ellas) y se clasifican mediante el clasificador de Naive Bayes, SVM y LDA. De esta forma, se comparan los resultados obtenidos con los distintos clasificadores sobre los datos imputados mediante los cuatro métodos de imputación, con los obtenidos al hacer la clasificación con los datos completos reales, y con los datos sin realizar ningún tipo de imputación. Esto permite para este caso concreto, hacer una comparativa entre los métodos de imputación y cuál ha sido el impacto en los métodos de clasificación utilizados.

Índice general

Capítulo 1

Introducción a la minería de datos

1.1. Minería de datos

"La minería de datos es el arte y la ciencia del análisis inteligente de datos." (Williams, 2011)

La minería de datos puede definirse inicialmente como un proceso de descubrimiento de nuevas y significativas relaciones, patrones y tendencias al examinar grandes cantidades de datos. (Pérez López y Santín González, 2008)

La disponibilidad de grandes volúmenes de información y el uso generalizado de herramientas informáticas ha transformado el análisis de datos orientándolo hacia determinadas técnicas especializadas englobadas bajo el nombre de minería de datos o *Data Mining*. (Pérez López y Santin González, 2008)

Las técnicas de minería de datos persiguen el descubrimiento automático del conocimiento contenido en la información almacenada de modo ordenado en grandes bases de datos. Estas técnicas se basan en el análisis de los datos utilizando tecnologías de reconocimiento de patrones, redes neuronales, lógica difusa, algoritmos genéticos y otras técnicas avanzadas de análisis de datos para lograr su objetivo. (Pérez López y Santin González, 2008)

Es un término relativamente moderno que integra numerosas técnicas de análisis de datos y extracción de modelos. Aunque se basa en varias disciplinas, algunas de ellas más tradicionales, lo que la diferencia de ellas es la orientación hacia el fin más que hacia el medio, hecho que permite beneficiarse de todas ellas sin prejuicios. La finalidad, como hemos mencionado, es ser capaces de extraer patrones, de describir tendencias y regularidades, de predecir comportamientos y, en general, de sacar partido a la información almacenada que nos rodea hoy en día, generalmente heterogénea y en grandes cantidades, permite a los individuos y a las organizaciones comprender y modelar de una manera más eficiente

y precisa del contexto en el que deben actuar y tomar decisiones. (Orallo et al., 2004)

El origen del término está en otros de la década de 1960, cuando en Estadística se hacía referencia a términos como *data fishing* (buscar o "pescar" en los datos), *data archaeology* o *data dredging* (dragado de datos) para referirse al análisis de grandes volúmenes de datos. No fue hasta el final de la década de 1980 y comienzo de la década de 1990 que se consolidó el término "minería de datos" y *Knowledge Data Discovery (KDD)*, en español, "descubrimiento del conocimiento en bases de datos". En esa época, sólo existían un par de empresas dedicadas a esta tecnología. Aparece, por tanto, el término *data mining* (minería de datos) con el que se conoce actualmente al proceso de obtención de conocimiento a partir de los datos por medio de su análisis. La aparición de la minería de datos como disciplina ha tenido mucho que ver con el cambio de concepción de los datos, unido a la gran cantidad de estos que se generan y almacenan continuamente en cualquier ámbito. Otro factor que también ha favorecido la consolidación de la minería de datos como disciplina es el gran avance en los últimos tiempos en las prestaciones y capacidad computacional. (Carmona et al., 2022)

Tradicionalmente, las técnicas de minería de datos se aplicaban sobre información contenida en almacenes de datos. No obstante, actualmente está cobrando una importancia cada vez mayor la minería de datos desestructurados como es la información contenida en ficheros de texto (*text mining*), en Internet (*web mining*), etc. Además, hoy en día han surgido otras necesidades de tipo operativo, como la integración de los resultados obtenidos en los sistemas de información en línea, con la exigencia, por tanto, de que los procesos funcionen prácticamente en tiempo real; por ejemplo, la alerta temprana frente a alarmas en una cadena de montaje, la detección instantánea del fraude en operaciones bancarias, un sistema de recomendación de productos en una tienda en línea, etc. (Carmona et al., 2022)

Los factores, pues, que nos han llevado a la aparición y desarrollo de esta disciplina en auge son los siguientes cuatro:

- El abaratamiento de los sistemas de almacenamiento tanto temporal como permanente.
- El incremento de las velocidades de cómputo en los procesadores.
- Las mejoras en la confiabilidad y aumento de la velocidad en la transmisión de datos.
- El desarrollo de sistemas administradores de bases de datos más poderosos.

Todas estas ventajas están haciendo que en la actualidad se abuse del almacenamiento

de gran volumen de información en cualquier ámbito, que bien analizada, puede proporcionar en conjunto un verdadero conocimiento de gran ayuda en la toma de decisiones.

El *data mining* trabaja buscando patrones, comportamientos, agrupaciones, secuencias, tendencias o asociaciones que puedan generar algún modelo que nos permita comprender mejor el dominio para ayudar en una posible toma de decisión.

Con todo lo anterior podemos decir que *Data Mining* es el proceso de descubrir patrones de información interesantes y potencialmente útiles, inmersos en una gran base de datos en la que se interactúa constantemente. *Data Mining* es una combinación de procesos como extracción de datos, limpieza de datos, selección de características, algoritmos y análisis de resultados.

Una definición tradicional es la siguiente: "Un proceso no trivial de identificación válida, novedosa, potencialmente útil y entendible de patrones comprensibles que se encuentran ocultos en los datos" (Fayyad et al., 1996). Desde otro punto de vista, de Molina et al. (2001), se define como "la integración de un conjunto de áreas que tienen como propósito la identificación de un conocimiento obtenido a partir de las bases de datos que aporten un sesgo hacia la toma de decisiones".

La minería de datos no surge como un área completamente nueva, sino más bien como la mezcla de conceptos procedentes de otras muchas y diferentes disciplinas:

- Estadística: muchas de las técnicas que se aplican en la minería de datos son o tienen su raíz en la Estadística. Se podría decir que la Estadística es la madre de la minería de datos.
- Bases de datos: el proceso de *KDD* parte de datos que, habitualmente, se encuentran almacenados en bases de datos.
- Visualización: el objetivo final de la minería de datos es obtener conocimiento que sea útil. Para lograrlo, es un requisito fundamental que ese conocimiento pueda ser visualizado por los expertos de cada dominio. De ahí la importancia de las técnicas de visualización (diagramas, gráficos, resúmenes, etc.)
- Aprendizaje automático: se encuentra profundamente ligado con la minería de datos, ya que ambos, de alguna manera, persiguen la obtención de modelos por medio de mecanismos automáticos.
- Otras: sistemas de apoyo a la decisión, recuperación de información, procesamiento de señales, etc.

La utilidad de la minería de datos se puede dar dentro de los aspectos siguientes:

- Sistemas parcialmente desconocidos: En estos casos habrá una parte del sistema que es conocida y habrá una parte aparentemente de naturaleza aleatoria. Bajo ciertas circunstancias, a partir de una gran cantidad de datos asociada con el sistema, existe la posibilidad de encontrar nuevos aspectos previamente desconocidos del modelo.

- Enorme cantidad de datos: al contar con mucha información, es importante encontrar la forma de analizarla y que ello produzca algún tipo de beneficio.

- Potente *hardware* y *software*: la capacidad y disponibilidad computacional ha aumentado considerablemente el desempeño del proceso de buscar y analizar información (el cual a veces debe vérselas con producciones de datos del orden de los *Gbytes*/hora)

Esta disciplina también tiene sus correspondientes riesgos y desafíos. Las mayores preocupaciones sobre la aplicación de la minería de datos son la seguridad y la privacidad, debido al uso de información potencialmente sensible o de identificación personal. (Ballesteros et al., 2018)

Los datos que se extraen deben ser completos, precisos y confiables ya que su finalidad es la toma de decisiones comerciales importantes y, a menudo, la interacción con el público, reguladores, inversores y socios comerciales. Las formas modernas de datos también requieren nuevos tipos de tecnologías, como reunir conjuntos de datos de una variedad de entornos informáticos distribuidos (también conocido como integración de *big data*) y para datos más complejos, como imágenes y video, datos temporales y datos espaciales. (Ballesteros et al., 2018)

1.1.1. El proceso de *KDD*

A principios de los años ochenta, Rakesh Agrawal, Gio Wiederhold, Robert Blum y Gregory Piatetsky-Shapiro, entre otros, empezaron a consolidar los términos de *data mining* y *KDD* o descubrimiento de conocimiento en bases de datos (en inglés, *Knowledge Discovery in Databases*, más conocido como proceso de *KDD*). Los términos de minería de datos y de *KDD* son usualmente confundidos, la realidad es que la minería de datos es una etapa del proceso *KDD*, la más importante.

Según la definición comúnmente aceptada, el proceso de *KDD* busca la extracción automatizada de conocimiento no trivial, implícito, previamente desconocido y potencialmente útil a partir de grandes volúmenes de datos.

Tras la preparación, los datos pasan a la fase de minería de datos, en la que se aplican una serie de técnicas para obtener modelos (representaciones simbólicas de la realidad que representan los datos de entrada) para, finalmente, ser validados e interpretados y así

poder obtener de ellos el ansiado conocimiento. Es éste un proceso iterativo, no lineal, que se retroalimenta.

El *data mining* es una tecnología compuesta por etapas que integra varias áreas y que no se debe confundir con un gran *software*. Durante el desarrollo de un proyecto de este tipo se usan diferentes aplicaciones en cada etapa que pueden ser estadísticas, de visualización de datos o de inteligencia artificial, principalmente. Actualmente existen aplicaciones o herramientas comerciales de *data mining* muy poderosas que facilitan el desarrollo de un proyecto. Sin embargo, casi siempre acaban complementándose con otra herramienta. (Carmona et al., 2022)

Las cuatro características que ha de poseer el conocimiento extraído por el proceso de *KDD* son:

- No trivial. De nada sirve extraer conocimiento conocido por todos o que carezca de importancia.
- Implícito. Se encuentra oculto en los datos.
- Previamente desconocido. Nada nuevo aporta si el conocimiento extraído ya había sido descubierto anteriormente.
- Útil. El conocimiento extraído debe servir para algo, de lo contrario no tiene ningún sentido invertir esfuerzos en extraerlo.

Según Carmona et al. (2022) el proceso de *KDD* se compone de las siguientes fases:

1. Recopilación de datos. En esta fase, los datos, procedentes de diferentes fuentes, se integran en un mismo y único repositorio de datos, denominado almacén de datos, más conocido como *data warehouse*. El resultado final de esta fase es, precisamente, ese *data warehouse*. Los datos generalmente se encuentran en múltiples fuentes diseñadas según esquemas desnormalizados y guardan información en torno a hechos, cada uno de los cuales está caracterizado por una serie de dimensiones, esquema que se denomina modelo multidimensional. Para integrarlas en un mismo almacén, es necesario disponer de un proceso que lea los datos de las diferentes fuentes, los limpie y los adecúe a la estructura que tiene el *data warehouse* para su almacenamiento. Este tipo de proceso se lleva a cabo mediante un sistema conocido como sistema *ETL* (*Extraction-Transformation-Load*). Una vez almacenados los datos, es posible tener que volver a repetir el proceso *ETL* para integrar nuevas fuentes de datos.
2. Filtrado de los datos y selección de variables. El formato de los datos contenidos en la fuente de datos nunca es el idóneo, y la mayoría de las veces no es posible

utilizar ningún algoritmo de minería. En esta fase se realiza una selección de los datos integrados en el *data warehouse*, pues dichos datos pueden no estar limpios, contener atributos irrelevantes, etc. Y se limpian y transforman de cara a fases posteriores. Las técnicas de selección tienen por objeto filtrar aquellos datos que no son relevantes para el análisis posterior, filtrado que se puede realizar a varios niveles:

- Filtrado de atributos. Es posible que algunos de los atributos de los datos a analizar no sean de interés.
- Filtrado de registros. En ocasiones, el objetivo puede ser eliminar algunos de los registros almacenados y quedarse sólo con los relevantes, pues con un subconjunto menor de registros (muestra) se podría hacer un análisis igual de efectivo, pero mucho más eficiente desde el punto de vista computacional. En este caso, se suelen aplicar técnicas de muestreo, como muestreo aleatorio simple, aleatorio estratificado o muestreo de grupos.

Por su parte, las tareas de limpieza de datos van, normalmente, encaminadas a resolver dos problemas bastante habituales:

- La ausencia de valores. Es muy habitual que, para muchos de los registros analizados, falte cierta información (datos faltantes o *missing values*) que, en ocasiones, aportan información interesante. Ante esta situación, se pueden adoptar diferentes alternativas:
  - Pasar por alto el valor faltante y continuar con el análisis.
  - Filtrar toda la columna asociada a dicho atributo.
  - Filtrar el registro que contiene el valor faltante.
  - Asignar un valor al atributo en cuestión, mediante uno de los posibles procedimientos de imputación automática.
- La existencia de valores erróneos. También suele ser común encontrar valores que, claramente, son erróneos. Aunque existen diferentes técnicas para detectar este tipo de valores (especificación de *edits*), la realidad es que, la mayoría de las veces, se realiza mediante procedimientos artesanales "*ad-hoc*". Una vez localizados los valores erróneos, las opciones más comunes para su tratamiento son las siguientes:
  - Pasar por alto el valor erróneo y continuar con el análisis.
  - Filtrar toda la columna asociada al valor erróneo.
  - Filtrar el registro que contiene el valor erróneo.

- Reemplazar el valor erróneo por un valor correcto, seguramente mediante el uso de alguna técnica específica de predicción.

Algunos autores consideran la identificación de objetos atípicos como un problema de detección de valores erróneos. El resultado de esta fase es un subconjunto limpio y transformado de los datos sobre el que ya se puede aplicar las técnicas de *data mining* en la siguiente fase. Finalmente, las técnicas de transformación de datos ofrecen soluciones a problemas que se pueden presentar como que los datos se encuentren en un determinado formato no adecuado para el uso de los diferentes algoritmos del *data mining*, formateando los datos según se necesite. Existen múltiples técnicas de transformación de datos. Algunas de las más aplicadas son:

- Numerización. Consiste en transformar un atributo de tipo cualitativo en uno equivalente cuantitativo.
- Discretización. Consiste en transformar un atributo cuantitativo en uno cualitativo ordinal.
- Creación de características. Consiste en la creación de un nuevo atributo en los datos, normalmente calculado como función de otros atributos ya existentes.
- Normalización. Consiste en la transformación del rango de valores que toma un determinado atributo. El caso más común de normalización es la normalización lineal uniforme, que transforma los valores de un atributo a una escala uniforme en el intervalo [0,1], utilizando

$$\text{Valor normalizado} = \frac{\text{Valor inicial} - \text{Valor mínimo}}{\text{Valor máximo} - \text{Valor mínimo}}.$$

- Reducción de la dimensionalidad. Las técnicas de reducción de la dimensionalidad buscan reducir el número de atributos sobre los que realizar el análisis posterior. Para ello, existen múltiples técnicas, aunque quizá la más conocida es la técnica de análisis de componentes principales (PCA), que proyecta los atributos iniciales en un espacio de dimensión mucho menor, de forma que en los nuevos atributos recogen la mayor parte de la información relevante de los originales, pero con la ventaja adicional de que se eliminan las posibles redundancias y dependencias que había. Esto permite un análisis más eficiente de los datos en términos de coste computacional.

Mediante el preprocesado, se filtran los datos (se eliminan valores incorrectos, no válidos, desconocidos, etc.), se obtienen muestras de los mismos (mayor velocidad de

respuesta del proceso), o se reducen el número de valores posibles (mediante redondeo, agrupamiento, etc.) Aún después de haber sido preprocesados, se sigue teniendo una cantidad ingente de datos. La selección de características reduce el tamaño de los datos, eligiendo las variables más influyentes en el problema, sin apenas sacrificar la calidad del modelo de conocimiento obtenido del proceso de minería. Los métodos para la selección de características son dos:

- Los basados en la elección de los mejores atributos del problema.

- Los que buscan variables independientes mediante tests de sensibilidad, algoritmos de distancia o heurísticos.

3. Extracción de conocimiento (*Data mining*). El siguiente paso consiste en aplicar técnicas concretas de minería de datos para obtener modelos.

   Una vez seleccionados, limpiados y transformados los datos, se obtiene el denominado conjunto de datos minables, y el siguiente paso del proceso de *KDD* consiste en aplicar técnicas de *data mining* para obtener modelos que representen a dichos datos.

   En la etapa del *data mining* se pueden aplicar diferentes técnicas para resolver distintos tipos de problemas, a los que se les conoce aquí con el nombre de tareas, que se suelen clasificar dependiendo del tipo de modelo que son capaces de generar; a saber:

   - Tareas predictivas. Son aquellas que se utilizan para predecir el valor desconocido de uno o varios atributos para uno o varios registros de los datos minables. Entre ellas se encuentran, entre otras:
     - Clasificación. Consiste en encontrar un modelo que, aplicado a un nuevo ejemplo sin clasificar, lo clasifique dentro de un conjunto predefinido de clases. Normalmente, el atributo a predecir es de tipo cualitativo, y recibe el nombre de atributo de clase.
     - Regresión. Es similar a la de clasificación, con la diferencia de que, en este caso, el atributo es de tipo cuantitativo.
   - Tareas descriptivas. Son aquellas que generan modelos que, de alguna forma, describen los datos, sin llevar a cabo ningún tipo de predicción. Entre las más importantes se encuentran:
     - Clustering. Pretende dividir una población heterogénea de objetos en grupos homogéneos, denominados clústeres de forma que los objetos de cada grupo sean muy similares entre sí. También se denomina segmentación o agrupamiento.

- Asociación. Pretende encontrar reglas que muestran la relación que existe entre los distintos atributos de los datos analizados, denominadas reglas de asociación.
- Detección de atípicos. Consiste en encontrar objetos que, dentro de un conjunto, manifiesten características significativamente diferentes a las del resto de los objetos del conjunto.

Para abordar cada una de las tareas anteriores, existen numerosas técnicas o algoritmos; y además de las anteriores, existen muchas tareas específicas para tipos de datos no convencionales.

Mediante una técnica se obtiene un modelo de conocimiento, que representa patrones de comportamiento observados en los valores de las variables del problema o relaciones de asociación entre dichas variables. También pueden usarse varias técnicas a la vez para generar distintos modelos.

4. Interpretación y evaluación. Los modelos obtenidos en la fase anterior han de ser evaluados. Tras la obtención de los modelos de *data mining*, el último paso del proceso de *KDD* consiste en evaluar la calidad de dichos modelos y realizar una interpretación de los mismos para obtener el conocimiento buscado.

   La evaluación de los modelos obtenidos en la fase anterior es una tarea crucial. No todos los modelos obtenidos cumplirán con las características esperadas en ellos para poder obtener conocimiento. En particular, los modelos han de ser precisos, comprensibles e interesantes.

   Aunque dependerá de la tarea de data mining en cuestión, en general, para evaluar un modelo se suele utilizar un enfoque consistente en reservar un pequeño subconjunto de los datos (conjunto de prueba) que se utilizará para validar el modelo construido con el resto de los datos (conjunto de entrenamiento), denominado validación simple.

   Una técnica algo más avanzada, a la vez que más utilizada, es la técnica de validación cruzada *n-fold cross validation*. En este caso, para validar un modelo, se elige aleatoriamente el $n\,\%$ de los datos como conjunto de prueba y con el $(100 - n)\,\%$ restante, como conjunto de entrenamiento, se construye el modelo; y el proceso se repite n veces variando cada vez los conjuntos de prueba y entrenamiento. Un valor muy habitual para n es 10.

   En general, para cada tarea de *data mining* se utilizan unas métricas específicas que miden la calidad de los modelos obtenidos con ellas:

- Clasificación. Para evaluar un modelo de clasificación se mide su precisión predictiva o porcentaje de clasificaciones acertadas en el conjunto de prueba frente al total de las clasificaciones realizadas.
- Regresión. En este caso, la medida típica que se suele emplear es el error cuadrático medio.
- Reglas de asociación. En este caso se suele medir el porcentaje de las instancias que la regla predice correctamente.
- Clustering. La calidad de una segmentación de objetos en clústeres se suele medir por medio de la cohesión de los clústeres; es decir, mediante alguna métrica que, efectivamente, mida si los objetos de cada clúster son similares entre sí y diferentes a los objetos de otros clústeres.

Una vez conocida la calidad de los modelos, es necesario expresarlos en términos del área de aplicación, para lo que es importante contar con técnicas de visualización de dichos modelos para que sean comprendidos e interpretados por los expertos de cada dominio. De esa manera, los expertos serán capaces de contrastar los modelos obtenidos con su propia visión de la realidad y transformarlos en conocimiento que será utilizado y difundido para el avance del área en cuestión.

Según algunos autores, la fase de filtrado de los datos y selección de variables se puede descomponer en varias fases, si bien lo importante es saber que el objetivo de la misma es preparar los datos para poder realizar *data mining* con ellos en la fase siguiente.

Una característica importante del proceso de *KDD* es su naturaleza iterativa. Esto significa que es posible tener que aplicar varias veces el proceso de *KDD* hasta obtener el conocimiento deseado.

El concepto de *KDD* se ha desarrollado, y continúa desarrollándose, desde la intersección de la investigación de áreas tales como bases de datos, aprendizaje automático, reconocimiento de patrones, estadística, teoría de la información, inteligencia artificial, razonamiento con incertidumbre, visualización de datos y computación de altas prestaciones. (Riquelme et al., 2006)

1.1.2. Los datos

La información primaria de la minería de datos no es estrictamente la almacenada en las denominadas bases de datos relacionales (que también), sino la que se encuentra en un denominado sistema de almacenamiento desnormalizado. Y el proceso de *KDD* utiliza diferentes tipos de datos:

- Cuantitativos: cuyos valores representan magnitudes (discretos y continuos)
- Cualitativos: cuyos valores representan una categoría y no una cantidad (nominales y ordinales)
- Otros: necesitarán enfoques particulares
    - Series temporales: sucesiones de valores que representan la evolución de una determinada característica a lo largo de un periodo de tiempo, con mediciones, generalmente, a intervalos de tiempo regulares.
    - Datos espaciales: que representan la estructura espacial de algún objeto.
    - Datos multimedia: imágenes, videos, elementos de audio, etc.
    - Documentos: descripciones textuales de objetos como, por ejemplo, resúmenes, obras literarias completas, etc.
    - Datos procedentes de la web: información acerca de la estructura de los sitios web, de los patrones de navegación de los usuarios, etc.

1.1.3. Utilidad y aplicaciones

La minería de datos puede resultar útil en una gran cantidad de situaciones; algunos problemas específicos que puede resolver son los siguientes:

- Encontrar grupos o tipologías de objetos, lo que recibe el nombre de *clustering*.
- Analizar y obtener modelos que se puedan utilizar de cara al futuro, lo que se conoce como técnicas de clasificación (por ejemplo, con minería de datos, al analizar una tabla se podría obtener un conjunto de reglas que recomiendan o no la concesión de un crédito, etc)
- Detectar grupos de variables o valores que están asociados (por ejemplo: los clientes de un supermercado que compran pan y azúcar, habitualmente también compran leche, etc.), lo que se conoce como técnicas de asociación.

Por lo que respecta a las aplicaciones, en general, la minería de datos se puede aplicar y resulta útil en casi cualquier dominio, siempre y cuando haya una cantidad suficiente de datos de los que extraer conocimiento. La minería de datos podría abordar problemas de distintos dominios como negocios, banca y finanzas, compañías de seguros, supermercados, educación, medicina, biología, internet, gobierno, empresariales, ...

1.2. Métodos supervisados y no supervisados

En la actualidad, existen diversos campos que se enfrentan al problema de disponer de bases de datos con un volumen de información almacenada que no puede ser analizada mediante técnicas estadísticas tradicionales. Como hemos comentado anteriormente, la extracción de conocimiento de los datos puede resultar de gran interés en diversos ámbitos, tanto para llegar a un conocimiento profundo de la estructura de los datos, como para servir de base en la toma de decisiones. Por otra parte, estas bases de datos se encuentran, en muchas ocasiones, en un proceso de crecimiento continuo, lo cual hace más patente aún la necesidad del uso de técnicas de análisis automatizadas.

Las técnicas de minería de datos comprenden una serie de algoritmos que son capaces de obtener relaciones entre distintos atributos. Pretenden obtener patrones o modelos a partir de los datos recopilados. En este sentido, los datos recogen un conjunto de hechos (una base de datos) y los patrones son expresiones que describen un subconjunto de los datos (un modelo aplicable a ese subconjunto).

La elección de la técnica viene determinada básicamente por dos condicionantes: el tipo de los datos y el objetivo que se quiera obtener. También es importante en esta elección el nivel de comprensibilidad que se quiera obtener del modelo final, ya que hay modelos fáciles de "explicar" al usuario como, por ejemplo, las reglas de asociación, y otros que entrañan claras dificultades como las redes neuronales o los vectores soporte. (Riquelme et al., 2006)

Las técnicas de minería de datos se clasifican en dos grandes categorías: supervisadas o predictivas y no supervisadas o descriptivas.

Cada técnica proporciona un enfoque para extraer la información de los datos y, en general, puede implementarse por varios algoritmos. Cada algoritmo representa la forma de desarrollar en la práctica una determinada técnica paso a paso. De esta forma, será necesario establecer en cada caso particular cual es la técnica más apropiada.

En el caso del aprendizaje supervisado existe un atributo especial normalmente denominado clase, presente en todos los ejemplos que especifica si el ejemplo pertenece o no a un cierto concepto, que será el objetivo del aprendizaje. Mediante una generalización del papel del atributo clase, cualquier atributo puede desempeñar ese papel, convirtiéndose la clasificación de los ejemplos según el atributo seleccionado en el objeto del aprendizaje.

Supongamos que partimos de un conjunto de ejemplos denominados de entrenamiento de un cierto dominio D, el objetivo del aprendizaje supervisado es construir criterios para determinar el valor del atributo clase en un ejemplo cualquiera del dominio. Estos criterios están basados en los valores de uno o varios de los otros pares (atributo, valor) que intervienen en la definición de los ejemplos.

Las supervisadas las conformarían las técnicas de:

- Regresión
  - Regresión (lineal simple, múltiple simple, polinomial, logística, ...)
  - ANOVA
  - MANOVA
- Clasificación
  - *Naive Bayes*
  - Árboles de desición
  - Redes neuronales
  - *K-NN* (condensado, reducido)
  - *Random forest*
  - *AdaBoost*
  - Análisis discriminante
  - SVM (máquinas de vectores de soporte)

El aprendizaje no supervisado aborda el aprendizaje sin supervisión. En este caso, se trata de ordenar los ejemplos en una jerarquía según las regularidades en la distribución de los pares atributo-valor sin la guía del atributo especial clase.

Entre estas técnicas (no supervisadas) podemos encontrar:

- Asociación
  - ACP
  - Análisis factorial
  - Escalamiento multidimensional
- Clustering
  - *K-meqns*
  - Análisis de correspondencias (simple y múltiple)
  - Análisis clúster (jerárquico y no jerárquico)

1.2.1. Aplicaciones

Tanto las técnicas supervisadas como las no supervisadas son utilizadas en muchos campos y tipos de análisis.

Se han publicado varios estudios relacionados con el análisis del rendimiento de los estudiantes en los exámenes mediante técnicas supervisadas. Por ejemplo, Kotsiantis et al. (2003) aplicaron diferentes técnicas de aprendizaje automático, como Naive Bayes, *k-Nearest Neighbours* (*k-NN*) para clasificar a los estudiantes en desertores y no desertores. Tanner y Toivonen (2010) realizaron un estudio en el que consideraron la aplicación del algoritmo *k-NN* para la clasificación de los alumnos que realizaban un curso de mecanografía táctil en línea en dos categorías, es decir, los que "abandonarían" y los que completarían el examen final. También analizaron la tarea de regresión en el contexto de la predicción de las puntuaciones reales de la prueba final. En Tomasevic et al. (2020) se realiza un análisis exhaustivo y una comparación de las técnicas más avanzadas de aprendizaje automático para resolver la tarea de predecir el rendimiento de los estudiantes en los exámenes. Debido a la necesidad de un conjunto de datos etiquetados para la interpretación de las muestras de datos analizadas, las técnicas de aprendizaje automático supervisado se consideran más adecuadas para dicha tarea de predicción que las no supervisadas. Se tuvieron en cuenta tres categorías diferentes de técnicas de aprendizaje automático supervisado: las basadas en similitudes, las basadas en modelos y las probabilísticas.

Otra aplicación de técnicas supervisadas se recoge en Majumder y Nath (2021), un estudio sobre diversas aplicaciones de la minería de datos y las técnicas de aprendizaje supervisado en la detección del fraude empresarial.

También se pueden realizar análisis de patrones de comidas mediante técnicas supervisadas de minería de datos, en Hearty y Gibney (2008), utilizan para ello redes neuronales artificiales y árboles de decisión.

En la detección de fraudes y abusos en la atención sanitaria se utilizan estas técnicas: árboles de decisión en Shinet al. (2012),y Liou et al. (2008); las redes neuronales en Liou et al. (2008), Ortega et al. (2006) y He et al. (1997); algoritmos genéticos en He et al. (1999); y máquinas de vectores soporte (SVM) en Kirlidog y Asuk (2012) y Kumar et al. (2010).

Por otro lado, las técnicas de minería de datos no supervisadas pueden servir para para agrupar bibliotecas en Indonesia, como se recoge en Rahim et al. (2021), un estudio para analizar la técnica de aprendizaje no supervisado en la realización de cluster maps del número de bibliotecas en los niveles educativos. En el que se utilizó la técnica de *k-medoids*.

En Barocio et al. (2019), se realiza la identificación de coherencia en línea y condiciones de estabilidad para grandes sistemas eléctricos interconectados mediante una técnica de minería de datos no supervisada. Este artículo presenta una novedosa técnica de minería de

datos en línea no supervisada para identificar grupos coherentes, detectar perturbaciones en el sistema eléctrico y determinar el estado de estabilidad del sistema. La parte innovadora del enfoque propuesto reside en combinar algoritmos simples tradicionales como la descomposición del valor singular (SVD) y *K-means* para la agrupación junto con un nuevo concepto basado en pendientes de agrupación. La combinación propuesta proporciona un valor añadido a otras aplicaciones que dependen de algoritmos similares disponibles.

También se realizan análisis de datos no supervisados en la minería de datos operativos de grandes edificios para la mejora de la eficiencia energética, como se recoge en Fan et al. (2018) ya que "a pesar de la potencia de los análisis supervisados en el modelado predictivo, los análisis no supervisados son más prácticos y prometedores a la hora de descubrir nuevos conocimientos a partir de un conocimiento previo limitado".

En el campo de la química, Zheng y Zhao (2020) proporcionan un nuevo método no supervisado de minería de datos basado en el autoencoder apilado para el diagnóstico de fallos en procesos químicos.

Estos ejemplos de aplicación, de entre todos los posibles que se podrían mencionar, destacan la gran utilidad de ambos tipos de técnicas de minería de datos, que se podrían implementar en casi cualquier campo que lleguemos a imaginar.

Capítulo 2

Métodos de tratamiento de falta de información

2.1. Introducción

En la actualidad, los datos faltantes en un estudio y/o análisis estadístico son un problema común y conocido. Los datos faltantes pueden influir en la fiabilidad de los resultados obtenidos, ya que pueden dar una imagen distorsionada de la realidad. Esto es especialmente importante cuando se trata de estudios científicos, donde la fiabilidad de los resultados es fundamental para garantizar la exactitud de los hallazgos. Este trabajo se va a centrar en el tratamiento de falta de información en minería de datos.

En estadística, los valores perdidos, o los datos ausentes, se dan cuando no hay un valor almacenado en la variable de una observación. Se trata de una situación frecuente. Muchos conjuntos de datos existentes, industriales y de investigación, contienen valores perdidos. Se presentan debido a diversas razones, como los procedimientos de introducción manual de datos, los errores de los equipos y las mediciones incorrectas. Por ello, es habitual encontrar datos ausentes en la mayoría de las fuentes de información utilizadas. La detección de datos incompletos es fácil en la mayoría de los casos, buscando valores nulos en un conjunto de datos. Sin embargo, esto no siempre es así, ya que los Valores Perdidos (o *missing values*) pueden aparecer en forma de valores atípicos o incluso de datos erróneos (es decir, fuera de los límites, *outliers*) (Luengo et al., 2012).

Los valores perdidos dificultan el análisis de los datos por parte de los analistas. Según Barnard y Meng (1999), hay tres tipos de problemas que suelen estar asociados a los valores perdidos:

1. Pérdida de eficacia.

2. Complicaciones en el manejo y análisis de los datos. Los programas computacionales los manejan en forma variable, lo que puede conducir a errores en los resultados y su interpretación. Esto se complica más aún si se adoptan diversas maneras de codificar los datos faltantes; si la estrategia incluye usar números reales para señalar un dato faltante el programa puede incluirlos en los posibles cálculos distorsionando en mayor o menor grado los resultados. Por otro lado, muchos programas computacionales simplemente omiten del análisis aquellos individuos que no tienen los datos completos. Esto reduce el tamaño muestral y los autores pueden no detectarlo pues el programa termina haciendo cálculos igual. No hay ninguna forma totalmente satisfactoria para el manejo de los datos faltantes, por lo que se debe ser estricto en optimizar la recolección y registro de datos en la etapa de diseño y ejecución (Danigno, 2014).

3. Sesgo resultante de las diferencias entre los datos que faltan y los completos. Si no se sabe nada sobre la o las causas por las cuales faltan datos es imposible descartar un posible sesgo y menos estimar su magnitud. Otro punto importante es la cantidad: si son pocos los datos faltantes, es probable que su efecto sea menor pero si son muchos su ausencia va comprometiendo progresivamente la validez de las conclusiones, es decir, el sesgo que introduce o puede introducir la falta de datos es proporcional al número de pérdidas. No hay una cifra establecida pero probablemente pérdidas mayores al 10 % no son aceptables en la mayoría de las circunstancias. (Danigno, 2014).

2.2. Tipos de datos faltantes

Es importante categorizar los mecanismos que conducen a la introducción de valores faltantes (Little y Rubin, 1987). Dicho mecanismo determinará el método para tratar su falta. Tal y como afirman Little y Rubin, existen tres mecanismos diferentes de inducción de datos perdidos:

- Falta completamente al azar (*Missing completely at random, MCAR*), cuando la distribución de un ejemplo que tiene un valor perdido para un atributo no depende ni de los datos observados ni de los datos perdidos (Little y Rubin, 1987). Otra forma de interpretarlo es que cualquier valor tiene la misma probabilidad de faltar que cualquier otro. Por ejemplo, fallos ocasionales de equipos que impiden hacer una medición, olvido ocasional en registrar un dato, el encargado de hacer la medición se enfermó o pérdidas de muestras porque se rompieron los tubos. Omitir del análisis a los individuos con datos faltantes no alteraría la validez pero podría disminuir la

potencia del estudio. Estimar a priori posibles pérdidas por este mecanismo debiera formar parte del protocolo en el cálculo del tamaño muestral. (Dagnino, 2014)

- Falta al azar (*Missing at random, MAR*), cuando la distribución de un ejemplo que tiene un valor perdido para un atributo depende de los datos observados, pero no depende de los datos perdidos (Little y Rubin, 1987), es decir, una o varias características registradas pueden explicar la distribución de datos faltantes. Por ejemplo: el nivel de respuestas faltantes en una encuesta está relacionado con el nivel socioeconómico, el número de pacientes con un ECG preoperatorio está relacionado con la edad de los pacientes, o un centro en un estudio multicéntrico no mide una variable particular porque no cuenta con los medios para ello. El nombre es confuso por lo que algunos prefieren "falta ignorable o manejable" estadísticamente. (Danigno, 2014)

- No faltante al azar (*Not missing at random, NMAR*), cuando la distribución de un ejemplo que tiene un valor faltante para un atributo depende de los valores faltantes (Little y Rubin, 1987). Los datos perdidos probablemente dependen o están relacionados con datos no observados. Por ejemplo: falta de respuesta en un cuestionario, pérdida durante el seguimiento. El sesgo o los sesgos que pueden introducirse son evidentes e invalidan en mayor o menor medida los resultados. (Danigno, 2014)

En el caso de la modalidad *MCAR*, el supuesto es que las distribuciones de los datos faltantes y completos son iguales, mientras que para la modalidad *MAR* son diferentes, y los datos faltantes pueden predecirse utilizando los datos completos (Little y Rubin, 1987). Como veremos más adelante, la mayoría de los métodos de imputación que existen hasta ahora asumen estos dos mecanismos.

2.3. Alternativas a realizar ante la presencia de datos faltantes

Existen una variedad de alternativas a considerar cuando se encuentran datos faltantes en un estudio y/o análisis estadístico. Estas alternativas pueden variar, dependiendo de la cantidad de datos que faltan y de la naturaleza de los datos faltantes.

Uno de los enfoques comunes es el de la imputación. Esta técnica consiste en usar los datos disponibles para estimar los datos faltantes. Esto se puede hacer de varias maneras, como la imputación promedio, la imputación de mediana, la imputación de la moda, la imputación de regresión, entre otros. Esta técnica puede ser útil cuando los datos faltantes son pequeños en cantidad y los datos disponibles son representativos.

Otra alternativa es la exclusión de los datos faltantes. Esta es una buena opción cuando la cantidad de datos faltantes es significativo y los datos disponibles no son representativos. Esto garantizará que los datos faltantes no influyan en los resultados.

Finalmente, es posible que se pueda recopilar los datos faltantes. Esta alternativa es útil cuando los datos faltantes son significativos y los datos disponibles son representativos. Esto permitirá a los investigadores obtener los datos necesarios para asegurar la exactitud de los resultados.

En Farhangfar et al. (2008) se resumen los tres principales enfoques para tratar los valores perdidos. La forma más sencilla de tratarlos, y que normalmente nos viene a la mente primero es elimiar los ejemplos que los contienen, pero, como acabamos de mencionar, este método sólo es práctico cuando los datos contienen un número relativamente pequeño de ejemplos con valores perdidos y cuando el análisis de los ejemplos completos no dará lugar a un sesgo grave durante la inferencia. Otro método consiste en convertir los valores que faltan en un nuevo valor (codificarlos en un nuevo valor numérico), pero se ha demostrado que este método tan simplista conduce a graves problemas de inferencia. Por otro lado, si un número significativo de ejemplos contiene valores perdidos para un número relativamente pequeño de atributos, puede ser beneficioso realizar la imputación (relleno) de los valores perdidos. (Luengo et al., 2010)

Vamos a centrarnos en estos últimos, los métodos de imputación. Su mayor ventaja es que el tratamiento de los datos perdidos es independiente del algoritmo de aprendizaje utilizado. Por ello, el usuario puede seleccionar el método más adecuado para cada situación a la que se enfrente. Existe una amplia familia de métodos de imputación, desde la imputación de medias hasta los que analizan las relaciones entre atributos. El mecanismo que conduce a la introducción de valores faltantes determinará qué método de imputación podría aplicarse, si es que se aplica. (Luengo et al., 2010)

2.4. Métodos de imputación

Los métodos de imputación sustituyen los valores perdidos por valores estimados basados en la información disponible en el conjunto de datos. Hay muchas opciones que varían desde métodos simplistas, como la imputación de la media, hasta métodos más robustos basados en las relaciones entre atributos. (Luengo et al., 2010)

Vamos a introducir y describir algunos métodos bastante utilizados.

- No imputar (*Do Not Impute, DNI*). Como su nombre indica, todos los datos que faltan permanecen sin reemplazar, por lo que las redes deben utilizar sus estrategias de valores perdidos por defecto.

- Eliminación de casos o ignorar faltantes (*Ignore Missing, IM*). Con este método, se descartan del conjunto de datos todas las instancias con al menos un valor perdido.

- Valor global más común del atributo para los atributos simbólicos y valor global medio para los atributos numéricos (*MC*) (Grzymala-Busse y Goodwin, 2005). Este método es muy sencillo: para los atributos nominales, el valor faltante se sustituye por el valor más común del atributo; los valores numéricos se sustituyen por la media de todos los valores del atributo correspondiente.

- Concepto de valor de atributo más común para los atributos simbólicos y concepto de valor medio para los atributos numéricos (*CMC*) (Grzymala-Busse y Goodwin, 2005). Como se indica en *MC*, sustituimos el valor perdido por el más repetido si es nominal o el valor medio si es numérico, pero considerando sólo las instancias con la misma clase que la instancia de referencia.

- Imputación con *K-Nearest Neighbor (KNNI)* (Batista y Monard, 2003). Con este algoritmo basado en instancias, cada vez que encontramos un valor faltante en una instancia actual, calculamos los k vecinos más cercanos e imputamos un valor a partir de ellos. Para los valores nominales, se toma el valor más común entre todos los vecinos, y para los valores numéricos utilizaremos el valor medio. Es necesario definir una distancia para medir la proximidad entre instancias, habitualmente se utiliza la distancia euclídea.

- Imputación ponderada con *K-Nearest Neighbor (WKNNI)* (Troyanskaya et al., 2001). El método *Weighted K-Nearest Neighbor* selecciona las instancias con valores similares (en términos de distancia) a una considerada, por lo que puede imputar como lo hace *KNNI*. Sin embargo, el valor estimado tiene ahora en cuenta las diferentes distancias a los vecinos, utilizando una media ponderada o el valor más repetido según la distancia.

- Imputación por agrupación de *K-means (KMI)* (Li et al., 2004). Dado un conjunto de objetos, el objetivo general de la agrupación es dividir el conjunto de datos en grupos basados en la similitud de los objetos, y minimizar la disimilitud intraclúster. En el *clustering* de *K-means*, la disimilitud intraclúster se mide sumando las distancias entre los objetos y el centroide del clúster (valor medio de los objetos del clúster) al que están asignados. Una vez que los clusters han convergido, el último proceso consiste en rellenar todos los atributos no referenciales de cada objeto incompleto basándose en la información del clúster. Los objetos de datos que pertenecen al mismo clúster se toman como vecinos más cercanos entre sí, y aplicamos un algoritmo

de vecino más cercano para reemplazar los datos que faltan, de forma similar a la imputación de vecinos más cercanos.

- Imputación con *Fuzzy K-means Clustering (FKMI)* (Acuña y Rodríguez, 2004; Li et al., 2004). En el *clustering* difuso, cada objeto de datos x_i tiene una función de pertenencia que describe el grado en que este objeto de datos pertenece a un determinado clúster v_k. En el proceso de actualización de las funciones de pertenencia y los centroides, sólo tenemos en cuenta los atributos completos. En este proceso, no podemos asignar el objeto de datos a un clúster concreto representado por un centroide de clúster (como se hace en el algoritmo básico de *clustering K-mean*), porque cada objeto de datos pertenece a todos los K clusters con diferentes grados de pertenencia. Reemplazamos los atributos de no referencia para cada objeto de datos incompleto x_i basándonos en la información sobre los grados de pertenencia y los valores de los centroides de los clústers.

- La Imputación de Máquinas de Vectores de Apoyo (*SVMI*) (Feng, Chen, Yin, Yang, y Chen, 2005) es un algoritmo basado en la regresión *SVM* para rellenar los datos que faltan, es decir, establecer los atributos de decisión (salida o clases) como los atributos de condición (atributos de entrada) y los atributos de condición como los atributos de decisión, por lo que podemos utilizar la regresión *SVM* para predecir los valores de atributos de condición que faltan. Para ello, primero seleccionamos los ejemplos en los que no faltan valores de atributos. En el siguiente paso establecemos uno de los atributos de condición (atributo de entrada), algunos de cuyos valores faltan, como el atributo de decisión (atributo de salida), y los atributos de decisión como los atributos de condición por contrarios. Por último, utilizamos la regresión *SVM* para predecir los valores de los atributos de decisión.

- *Event Covering (EC)* (Wong y Chiu, 1987). Basado en el trabajo de Wong et al., un modelo de probabilidad de modo mixto se aproxima por uno discreto. En primer lugar, discretizan los componentes continuos utilizando un criterio de mínima pérdida de información. Tratando una n-tupla de características de modo mixto como una de valores discretos, los autores proponen un nuevo enfoque estadístico para la síntesis del conocimiento basado en el análisis de conglomerados. Este método tiene la ventaja de no requerir la normalización de la escala ni la ordenación de los valores discretos. Por síntesis de los datos en conocimiento estadístico, se refieren a los siguientes procesos: (1) sintetizar y detectar a partir de los datos patrones inherentes que indican interdependencia estadística; (2) agrupar los datos dados en clusters inherentes basados en esta interdependencia detectada; y (3) interpretar los patro-

nes subyacentes para cada cluster identificado. El método de síntesis se basa en el enfoque de cobertura de eventos del autor. Con el método de inferencia desarrollado, podemos estimar los valores faltantes en los datos.

- Maximización de expectativas regularizada (EM) (Schneider, 2001). Los valores perdidos se imputan con un algoritmo de maximización de expectativas regularizado (EM). En una iteración del algoritmo EM, las estimaciones dadas de la media y de la matriz de covarianza se revisan en tres pasos. En primer lugar, para cada registro con valores perdidos, los parámetros de regresión de las variables con valores perdidos sobre las variables con valores disponibles se calculan a partir de las estimaciones de la media y de la matriz de covarianza. En segundo lugar, los valores que faltan en un registro se rellenan con sus valores de esperanza condicional dados los valores disponibles y las estimaciones de la media y de la matriz de covarianza, siendo los valores de esperanza condicional el producto de los valores disponibles y los coeficientes de regresión estimados. En tercer lugar, se reestiman la media y la matriz de covarianza, la media como la media muestral del conjunto de datos completado y la matriz de covarianza como la suma de la matriz de covarianza muestral del conjunto de datos completado y una estimación de la matriz de covarianza condicional del error de imputación. El algoritmo EM comienza con las estimaciones iniciales de la media y de la matriz de covarianza y recorre estos pasos hasta que los valores imputados y las estimaciones de la media y de la matriz de covarianza dejan de cambiar apreciablemente de una iteración a la siguiente.

- Imputación por descomposición de valores singulares ($SVDI$) (Troyanskaya et al., 2001). En este método, empleamos la descomposición de valores singulares para obtener un conjunto de patrones de expresión mutuamente ortogonales que pueden combinarse linealmente para aproximar los valores de todos los atributos del conjunto de datos. Para ello, primero calculamos los valores perdidos con el algoritmo EM y, a continuación, calculamos la descomposición del valor singular y obtenemos los valores propios. Ahora podemos utilizar los valores propios para aplicar una regresión sobre los atributos completos de la instancia, para obtener una estimación del propio valor faltante.

- Análisis Bayesiano de Componentes Principales ($BPCA$) (Oba et al., 2003). Este método es un método de estimación de los valores perdidos, que se basa en el análisis bayesiano de componentes principales. Aunque la metodología de que un modelo probabilístico y las variables latentes se estimen simultáneamente en el marco de la inferencia de Bayes no es nueva en principio, la implementación real del $BPCA$ que

permite estimar variables perdidas arbitrarias es nueva en términos de metodología estadística. El método de estimación de valores perdidos basado en el *BPCA* consta de tres procesos elementales. Se trata de (1) la regresión de componentes principales (*PC*), (2) la estimación bayesiana y (3) un algoritmo repetitivo similar a la maximización de esperanzas (*EM*).

- Imputación local por mínimos cuadrados (*LLSI*). (Kim et al., 2005). Con este método, una instancia objetivo que tiene valores perdidos se representa como una combinación lineal de instancias similares. En lugar de utilizar todos los valores disponibles en los datos, sólo se utilizan los valores similares basados en una medida de similitud el método tiene la connotación "local". Hay dos pasos en el *LLSI*. El primer paso es la selección de k valores mediante la norma *L2*. El segundo paso es la regresión y la estimación, independientemente de cómo se seleccionen los k valores. Los autores utilizan un método heurístico de selección de k parámetros.

- Correspondencia predictiva de medias (*Predictive Mean Matching (PMM)*) imputa un valor observado real, cuyo valor predicho se encuentra entre un conjunto de $k \geq 1$ valores, que son los más cercanos al predicho para el caso que falta. PMM suele ser más capaz de preservar la distribución original de los datos empíricos que los enfoques de imputación múltiple totalmente paramétricos, cuando los datos empíricos se desvían de sus supuestos de distribución. Por lo tanto, el uso de PMM es especialmente útil en situaciones en las que se violan los supuestos del modelo de los procedimientos de imputación múltiple totalmente paramétricos y en las que estos procedimientos producirían estimaciones muy poco plausibles. Desgraciadamente, hoy en día sólo existen unos pocos estudios que hayan probado sistemáticamente la robustez de PMM y todavía se desconoce en gran medida dónde se encuentran exactamente los límites de este procedimiento. (Kleinke, 2017)

2.5. Análisis de sensibilidad sobre los resultados

El análisis de sensibilidad es una herramienta importante para evaluar los resultados obtenidos al utilizar métodos de imputación para tratar los valores faltantes. Como, mencionamos antes, los datos faltantes tienen un impacto significativo en el análisis de los resultados. Los resultados obtenidos a partir de las técnicas de imputación pueden variar según el método utilizado. Por lo tanto, es importante evaluar los resultados obtenidos para comprender mejor la influencia de los valores faltantes en el análisis de los datos.

Este análisis se utiliza para examinar la sensibilidad de los resultados a los cambios en

los parámetros. Esto permite identificar los factores que tienen un impacto significativo en los resultados y permitir una toma de decisiones informada.

Permite valorar el efecto sobre los resultados de tipo de datos utilizados, los ausentes o los valores extremos, entre otros. Un análisis de sensibilidad determina cómo se ven afectadas las variables en función del resto.

Siempre que nos podamos preguntar si los resultados cambiarían si cambiásemos alguna de las definiciones del estudio, o el método de análisis, o el modo de tratar los datos faltantes (en nuestro caso) o el cumplimiento o violaciones del protocolo de estudio, la validez de nuestros resultados puede verse comprometida. Para defendernos de esto podemos realizar este análisis y si los resultados siguen siendo los mismos, podremos decir que nuestras conclusiones son robustas. (Molina Arias, 2013)

Un análisis de sensibilidad es, por tanto, el método que usamos para determinar la robustez de una valoración examinando en qué grado los resultados se influyen por cambios en la metodología o en los modelos utilizados en el estudio. (Molina Arias, 2013)

En el caso que nos concierne, el de los valores faltantes, tenemos dos opciones: hacer el análisis solo con los datos completos o imputar los valores que faltan para incluirlos todos en el análisis. Con ambas posibilidades corremos riesgo de sesgos, dependiendo en gran parte de qué condiciona que falten los datos y de si los datos que se pierden son al azar o no. Habitualmente se hace el análisis completo y el análisis con imputación de datos y se estudian las diferencias en los resultados obtenidos para hacer la valoración. (Molina Arias, 2013)

Entre las ventajas de realizar este tipo de análisis podemos encontrar las siguientes:

- Facilita la toma de decisiones: el análisis de sensibilidad puede resultar de gran utilidad para tomar una decisión, por lo que resulta muy ventajoso para los autores del estudio, sobre todo a la hora de planificar lo que se pretende llevar a cabo. Esto es así porque el análisis de sensibilidad dará como resultado diversos pronósticos sobre un estudio o análisis concreto que estarán fundamentados en datos.

- Asegura el control de calidad del estudio: gracias al análisis de sensibilidad, los expertos podrán determinar qué procesos y estudios no están dando los resultados esperados, es decir, qué estudios no cumplen con los objetivos que se fijaron en un principio. De esta forma, y gracias al análisis de sensibilidad, los expertos podrán detectar los errores y fallos que se están produciendo, lo cual les permitirá subsanarlos y esto redundará positivamente en la calidad del estudio y análisis, así como suponer un ahorro importante de tiempo.

- Mejora en la asignación de los recursos disponibles: gracias al análisis de sensibilidad,

los expertos podrán determinar cuáles son las fortalezas y las flaquezas de un proceso o estudio y, con base en esta información, podrán asignar de mejor manera los recursos de que disponen. De esta forma, el impacto de los recursos redundará en los resultados del estudio.

- Pronóstico del éxito o fracaso de un estudio: el análisis de sensibilidad arroja resultados fiables, ya que estos están basados en datos confiables y certeros. Así, al estudiar las diferentes variables y los eventuales resultados que pueden producirse, se podrán tomar mejores y más fundamentadas decisiones, lo que facilitará el éxito del estudio.

Se puede realizar con una variedad de métodos, incluidos los gráficos de sensibilidad, los análisis de factores principales, los análisis de regresión y los análisis de variación. Estos métodos se pueden utilizar para evaluar los resultados obtenidos al utilizar métodos de imputación. Como hemos mencionado, el análisis de sensibilidad se puede utilizar para examinar cómo los cambios en los parámetros de imputación afectan los resultados del análisis. Esto se puede utilizar para comprender mejor la influencia de los valores faltantes en los resultados del análisis.

A pesar de los beneficios del análisis de sensibilidad para evaluar los resultados obtenidos al utilizar métodos de imputación, hay desafíos que deben superarse. Quizá el inconveniente principal del análisis de sensibilidad es que este tan solo puede estudiar los cambios que se produce en una sola variable cada vez.

El tiempo requerido para realizarlo es otro de los principales desafíos. Puede ser un proceso complejo y costoso, ya que puede requerir una cantidad significativa de tiempo y recursos para realizar el análisis.

Otro desafío es la selección del método adecuado para el análisis de sensibilidad. Esto puede ser un reto, ya que hay una variedad de métodos disponibles para este análisis. Se deben considerar los objetivos de investigación, los datos disponibles y los recursos disponibles para determinar el método adecuado para el análisis.

Las métricas utilizadas en las técnicas supervisadas y no supervisadas son diferentes en algunos aspectos, pero hay algunas métricas que se usan en ambos casos. Estas incluyen sensibilidad, especificidad, curvas ROC, métrica de silueta y medidas de precisión y F1 *score*. Para las técnicas supervisadas normalmente se utilizan las siguientes:

- Matriz de confusión: es una tabla que describe el rendimiento de un modelo supervisado en los datos de prueba, donde se desconocen los verdaderos valores. Se llama así porque hace que sea fácil detectar dónde el sistema está confundiendo dos clases. Contiene

  - Verdaderos positivos (VP): cuando la clase real del punto de datos era 1 (Verdadero) y la predicha es también 1 (Verdadero)
  - Verdaderos negativos (VN): cuando la clase real del punto de datos fue 0 (Falso) y el pronosticado también es 0 (Falso).
  - Falsos positivos (FP): cuando la clase real del punto de datos era 0 (Falso) y el pronosticado es 1 (Verdadero).
  - Falsos Negativos (FN): Cuando la clase real del punto de datos era 1 (Verdadero) y el valor predicho es 0 (Falso).

- Exactitud o *accuracy*: la fracción de predicciones que el modelo realizó correctamente. Se representa como un porcentaje o un valor entre 0 y 1. Es una buena métrica cuando tenemos un conjunto de datos balanceado, es decir, cuando el número de observaciones de cada clase es similar.

- Sensibilidad indica la proporción de ejemplos positivos que están identificados correctamente por el modelo entre todos los positivos reales. Es decir,

$$\frac{VP}{(VP + FN)}$$

- Precisión: esta métrica está determinada por la fracción de elementos clasificados correctamente como positivo entre todos los que el modelo ha clasificado como positivos. La fórmula es

$$\frac{VP}{(VP + FP)}$$

- F1 *score*: combina las métricas Precision y Sensibilidad para dar un único resultado. Esta métrica es la más apropiada cuando tenemos conjuntos de datos no balanceados. Se calcula como la media armónica de Precision y Sensibilidad. La fórmula es

$$F1 = (2 \cdot Precision \cdot Sensibilidad)/(Precision + Sensibilidad)$$

  Se utiliza la media armónica y no la simple porque la media armónica hace que si una de las dos medidas es pequeña (aunque la otra sea máxima), el valor de F1 score va a ser pequeño.

- Especificidad: también conocida como verdadero negativo, mide la proporción de predicciones negativas que fueron correctas,

$$\frac{VN}{VN + FP}$$

- La curva ROC: Medir el área bajo la curva ROC es también un método muy útil para evaluar un modelo. Al trazar la tasa positiva verdadera (sensibilidad) frente a la tasa de falsos positivos (1 - especificidad), obtenemos la curva de Característica Operativa del Receptor (ROC). Esta curva nos permite visualizar el equilibrio entre la tasa de verdaderos positivos y la tasa falsos positivos.

Estas métricas se usan para medir la precisión de los resultados obtenidos al entrenar un modelo con los datos.

En las técnicas no supervisadas, las métricas más comunes son la medida de silueta, la métrica de Davies-Bouldin y la distancia entre clusters. Estas métricas se usan para evaluar la calidad de los resultados producidos al aplicar algoritmos de agrupamiento y clustering.

- *Silhouette* se refiere a un método de interpretación y validación de la coherencia dentro del análisis de grupos. La técnica proporciona una representación gráfica sucinta de lo bien que se ha clasificado cada objeto. El valor de la silueta es una medida de cuán similar es un objeto a su propio cúmulo (cohesión) en comparación con otros cúmulos (separación). La silueta va de -1 a +1, donde un valor alto indica que el objeto está bien emparejado con su propio cúmulo y mal emparejado con los cúmulos vecinos. Si la mayoría de los objetos tienen un valor alto, entonces la configuración del cúmulo es apropiada. Si muchos puntos tienen un valor bajo o negativo, entonces la configuración de cúmulos puede tener demasiados o muy pocos cúmulos.

- Índice de Davies-Bouldin: es una métrica para evaluar algoritmos de agrupamiento. Valores pequeños para el índice DB indica clústeres compactos, y cuyos centros estas bien separados los unos de los otros. Consecuentemente el número de clústeres que minimiza el índice DB se toma como el óptimo.

2.6. Métodos de tratamiento de falta de información en minería de datos

El impacto de los datos perdidos puede variar significativamente según el algoritmo de minería de datos que se utilice. Por ejemplo, un algoritmo de minería de datos basado en reglas puede ser mucho más sensible a la pérdida de datos que un algoritmo basado en árboles. Esto se debe a que los algoritmos basados en reglas se basan en la identificación de patrones repetidos en los datos, lo que significa que incluso una pequeña pérdida de datos puede dar lugar a un resultado muy diferente. Por otro lado, los algoritmos basados en árboles son mucho menos sensibles a la pérdida de datos, ya que se basan en patrones

más generales y abstracciones. Por lo tanto, el impacto de la pérdida de datos dependerá en gran medida del algoritmo de minería de datos que se utilice.

2.6.1. El impacto de la ausencia de datos en el algoritmo de *k-NN*

En el algoritmo k-vecino próximo más cercano, la propia naturaleza del algoritmo se basa en la precisión de los datos. Los datos que faltan y los imprecisos tienen un grave impacto en el rendimiento de este tipo de algoritmo. Si faltan datos por completo, pueden producirse clusters (distribuciones de datos) erróneos en función de la frecuencia y la categorización de los casos que contienen los datos que faltan. Un método para ayudar a resolver este problema es utilizar el propio algoritmo de minería de datos *k-nearest neighbor* para abordar el problema de los datos que faltan. Los valores imputados obtenidos pueden utilizarse para mejorar el rendimiento del propio algoritmo del vecino más próximo. (Brown y Kros, 2003)

En primer lugar, se identifican los k vecinos más próximos (los que no contienen datos que faltan) a la observación que sí contiene datos que faltan. La k es una constante predeterminada que representa el número de vecinos que no contienen datos que faltan y que deben tenerse en cuenta en el análisis. Según Witten y Frank (2000), se aconseja que el valor de k sea pequeño, por ejemplo cinco, para que el impacto de cualquier ruido presente sea mínimo. (Brown y Kros, 2003)

Por lo tanto, este algoritmo no se recomienda para grandes conjuntos de datos (Adriaans y Zantinge, 1997). Una vez identificados estos "vecinos", se puede asignar la clase mayoritaria para el atributo en cuestión al caso que contiene el valor que falta. Berson et al. (2000) sostienen que una base de datos histórica que contenga atributos con valores predictores similares a los del caso infractor también puede utilizarse para ayudar en la clasificación de registros no clasificados. (Brown y Kros, 2003)

Los inconvenientes de infravaloración de la varianza, distorsión de la distribución y decrecimiento de la correlación deben abordarse siempre que se utilice un valor constante para sustituir los datos que faltan. La proporción de valores sustituidos debe calcularse y compararse con todos los conglomerados e identificación de categorías que existían antes de la sustitución de los datos que faltan. (Brown y Kros, 2003)

2.6.2. El impacto de la ausencia de datos en árboles de decisión

Los árboles de decisión son una buena metodología para tratar los datos que faltan cuando se producen con frecuencia (Berry y Linoff, 1997). Los árboles de decisión también se adaptan muy bien a grandes conjuntos de datos (Adriaans y Zantinge, 1997). A veces es útil podar el árbol cuando hay un exceso de datos que faltan en determinadas ramas

(Berry y Linoff, 1997). La eliminación de determinados caminos puede ser necesaria para garantizar que el éxito general del proceso de toma de decisiones no se vea inhibido por la inclusión de casos que contienen datos que faltan. Witten y Frank (2000) aconsejan el uso de la prepoda durante el proceso de construcción del árbol para determinar cuándo dejar de desarrollar subárboles. La postpoda puede utilizarse una vez que el árbol está completamente construido. Si se opta por la poda posterior, las decisiones sobre las reglas de poda pueden tomarse después de haber construido y analizado el árbol. (Brown y Kros, 2003)

2.6.3. El impacto de la ausencia de datos en las normas de asociación

Como se recoge en Brown y Kros (2003), las reglas de asociación ayudan a identificar cómo se relacionan varios valores de atributos dentro de un conjunto de datos. Dado que las reglas de asociación se desarrollan muchas veces para ayudar a identificar diversas regularidades (patrones) dentro de un conjunto de datos, se ha descubierto que los algoritmos que utilizan reglas de asociación funcionan mejor con grandes conjuntos de datos. Se desarrollan para predecir el valor de un atributo (o conjuntos de atributos) en el mismo conjunto de datos (Darling, 1997). El objetivo principal del descubrimiento de reglas de asociación es identificar reglas que se apliquen a un gran número de casos con los que las reglas puedan relacionarse directamente, los datos que faltan pueden sobreestimar tanto el apoyo como la confianza de cualquier conjunto de reglas recién descubierto (Witten y Frank, 2000).

Los atributos que contienen valores de datos que faltan o están corruptos pueden dar lugar fácilmente a la creación de conjuntos no válidos de reglas o al fracaso en la identificación de patrones válidos que normalmente existen en los datos. Sin embargo, si el conjunto de datos utilizado para entrenar el algoritmo sólo contiene datos "inmaculados", suele producirse un sobreajuste del modelo basado en los patrones incluidos en el conjunto de entrenamiento. (Brown y Kros, 2003)

Por lo tanto, es necesario desarrollar reglas para las "excepciones a los conjuntos de reglas" que se han construido violando los datos correctos o "limpios". A continuación, es necesario poblar el conjunto de entrenamiento de los algoritmos que utilizan reglas de asociación con un porcentaje suficiente de "datos ruidosos", que representen todos los tipos posibles de excepciones a las reglas existentes. (Brown y Kros, 2003)

De este modo, se pueden desarrollar reglas de excepción para manejar todos los patrones de ruido que puedan asociarse a un conjunto de datos determinado, en lugar de rediseñar conjuntos de reglas que traten datos "limpios" o intentar introducir en esos conjuntos casos que no pertenezcan a los conjuntos de reglas existentes. A medida que se descubren

excepciones para las excepciones iniciales, se crea un tipo de estructura de árbol que forma una lista de decisiones para el tratamiento de los datos ausentes y ruidosos del conjunto de datos. Es necesario utilizar tanto reglas proposicionales como reglas relacionales en el conjunto de reglas para el tratamiento de los datos ausentes o ruidosos. (Brown y Kros, 2003)

Las reglas proposicionales contrastan el valor de un atributo con un valor constante, desarrollando así límites muy concisos para distinguir entre datos "limpios" y "ruidosos". En casos extremos, las constantes, los puntos de ruptura y los valores de los atributos asociados se utilizan para hacer crecer un árbol de regresión con el fin de estimar los valores de los datos que faltan en diversas condiciones. (Brown y Kros, 2003)

La incorporación de una regla o conjunto de reglas adicionales para tratar las excepciones (como los datos que faltan) puede ser fácil, ya que algunas reglas pueden desarrollarse para predecir múltiples resultados. (Brown y Kros, 2003)

Aunque una regla pueda tener tanto un alto nivel de apoyo como de confianza, una evaluación subjetiva por parte del usuario final puede determinar el grado de interés de una regla recién descubierta (Groth, 2000). Algunos paquetes de software de reglas de asociación pueden estar capacitados para eliminar automáticamente las "reglas poco interesantes". Por lo tanto, deben establecerse valores mínimos (puntos de ruptura) tanto para la confianza como para el apoyo de las reglas recién descubiertas. (Brown y Kros, 2003)

En algunos casos, puede establecerse una jerarquía de reglas, de modo que unas reglas impliquen a otras. En algunos casos, sólo la regla más fuerte se presenta como regla recién descubierta y las reglas de "menor fuerza" (apoyo y confianza) se vinculan a la regla más fuerte para su uso posterior (Han y Kamber, 2001).

2.6.4. El impacto de la ausencia de datos en las redes neuronales

Las redes neuronales han demostrado ser fiables y eficaces en aplicaciones de predicción, clasificación y agrupación (Adriaans y Zantinge, 1997). Los datos faltantes tienen un impacto similar en las redes neuronales que en otros tipos de algoritmos de clasificación, como k-vecino más próximo. Estas similitudes incluyen la subestimación de la varianza, la distorsión de la distribución y la decrecimiento de la correlación. (Brown y Kros, 2003)

Cuando se utilizan redes neuronales con datos que faltan en el proceso de extracción de datos, puede ser necesario "entrenar" la red inicial con datos que faltan si los datos que se van a probar y evaluar posteriormente van a contener datos que faltan. Al entrenar la red sólo con datos "limpios", los pesos internos desarrollados con el conjunto de entrenamiento no pueden aplicarse con precisión al conjunto de prueba. (Brown y Kros, 2003)

Una pregunta habitual que se plantea es "¿Cómo afecta la falta de datos a la ejecución

interna de la red neuronal? Dado que las ponderaciones internas utilizadas para calcular los resultados se crean y distribuyen dentro de la red sin proporcionar la perspectiva de cómo se crea una solución, los datos ausentes o sucios pueden distorsionar las ponderaciones que se asignan como las asociaciones entre nodos de una manera desconocida para el analista investigador. (Brown y Kros, 2003)

Mientras que la capa oculta es donde se desarrollan los pesos reales de la red, la función de activación combina las entradas de la red en una única salida (Westphal y Blaxton, 1998). La salida permanece baja hasta que las entradas combinadas alcanzan un umbral predeterminado, y pequeños cambios en la entrada pueden tener un efecto importante en la salida (Groth, 2000). La función de activación puede ser muy sensible a la falta de datos. (Brown y Kros, 2003)

La función de activación de la unidad básica de una red neuronal tiene dos subfunciones: la función de combinación y la función de transferencia. La función de combinación suele utilizar la "suma ponderada estándar"(la suma de los valores de los atributos de entrada multiplicada por los pesos asignados a dichos atributos) para calcular un valor que se transmite a la función de transferencia. (Brown y Kros, 2003)

La función de transferencia aplica una función lineal o no lineal al valor que le transmite la función de combinación. Aunque una función lineal utilizada en una red neuronal *feed-forward* simplemente realiza una regresión lineal, los valores faltantes pueden distorsionar los coeficientes de la ecuación de regresión y, por lo tanto, transmitir valores no válidos como salida (Berry y Linoff, 1997).

Capítulo 3

Aplicación práctica

3.1. Procedimiento

Se han seleccionado dos conjuntos de datos para ejemplificar lo tratado a lo largo de este trabajo. Estos dos conjuntos de datos no contienen ningún valor faltante por lo que se han extraído algunos para poder imputarlos, cuyo proceso se menciona más adelante.

3.1.1. Imputación de los datos faltantes

Una vez obtenidos los conjuntos de datos con los datos faltantes, se pasa a la imputación de estos. La imputación se va a realizar mediante cuatro métodos diferentes, y después se hará una comparación del acierto de imputación que ofrecen. En primer lugar se imputarán los datos mediante la técnica del valor global más común del atributo para los atributos simbólicos y valor global medio para los atributos numéricos (MC). Después se imputarán los datos faltantes mediante la técnica de *K-Nearest Neighbors (KNN)*, *Weighted K-Nearest Neighbor (WKNN)*, y, por último, mediante *Predictive Mean Matching Imputation.*

Se han elegido estos cuatro métodos de imputación por su gran utilización y su facilidad de implementación en R.

Valor global más común del atributo para los atributos simbólicos y valor global medio para los atributos numéricos

Este tipo de imputación es un método rápido en el cual se reemplazan los valores faltantes por la media de la variable en la cual se encuentren estos valores, en el caso de variables numéricas, o por el valor más común de la variable, en el caso de las variables simbólicas. Su facilidad de implementación se contrapone con los grandes inconvenientes que presenta esta técnica: subestimará la varianza, alterará las relaciones entre las variables,

sesgará casi cualquier estimación que no sea la media y sesgará la estimación de la media cuando los datos no sean *MCAR*.

K-Nearest Neighbor

El método de imputación con *K-Nearest Neighbor*, también conocido como *K-NN*, es una técnica de imputación de valores faltantes que utiliza una medida de similitud para encontrar los k registros más similares en un conjunto de datos y utiliza esos valores para imputar el valor faltante.

Para utilizar el método *K-NN*, primero se debe elegir el número k de vecinos más cercanos a considerar. Una vez que se ha seleccionado k, se buscan los k registros más similares en el conjunto de datos. Esto se puede hacer utilizando una medida de similitud, habitualmente la distancia Euclidiana.

Una vez que se han encontrado los k vecinos más cercanos, se pueden utilizar diferentes métodos para imputar el valor faltante. Una opción es simplemente tomar el valor medio de los k vecinos. Otra opción es utilizar un peso basado en la similitud de cada vecino, de modo que los vecinos más similares tengan un mayor peso en la imputación del valor faltante, como veremos en *Weighted K-Nearest Neighbor*.

Uno de los principales beneficios del método *K-NN* es que es fácil de implementar y no requiere ningún supuesto sobre la distribución de los datos. Además, no es necesario realizar ninguna transformación de los datos antes de utilizar el método *K-NN*. Sin embargo, el método *K-NN* tiene algunas desventajas. En primer lugar, puede ser computacionalmente costoso, ya que se necesita calcular la similitud entre todos los pares de registros en el conjunto de datos. En segundo lugar, el método *K-NN* es sensible al ruido y a los valores atípicos en los datos. Esto significa que un solo valor atípico o ruidoso puede tener un gran impacto en la imputación del valor faltante.

A pesar de estas desventajas, el método *K-NN* sigue siendo una técnica popular y ampliamente utilizada para imputar valores faltantes en los datos. Es especialmente útil en casos en que se dispone de una gran cantidad de datos y se desea utilizar toda la información disponible para imputar los valores faltantes de manera precisa.

Para implementar esta técnica vamos a utilizar la función de R *kNN* de la librería *vim*. Esta función establece el valor faltante sobre la media de todos los valores situados a k-distancias de la celda que contiene el faltante. Utiliza una distancia euclideana para determinar el valor faltante.

Como se ha mencionado, cuando se utiliza *k-NN*, hay que tener en cuenta muchos parámetros. En cuanto al número de vecinos a buscar, tomar un k bajo aumentará la influencia del ruido y los resultados van a ser menos generalizables. Por otro lado, tomar un

k alto tenderá a difuminar los efectos locales, que son exactamente lo que estamos buscando. También se recomienda tomar una k impar para las clases binarias para evitar los empates. En cuanto al método de agregación a utilizar, aquí se establece la media aritmética, la mediana y la moda para las variables numéricas y la moda para las categóricas.

Weighted K-Nearest Neighbor

Como mencionamos anteriormente, esta técnica es una modificación de la anterior, donde se tiene en cuenta un peso basado en la similitud de cada vecino, de modo que los vecinos más similares tengan un mayor peso en la imputación del valor faltante.

El valor estimado tiene ahora en cuenta las diferentes distancias a los vecinos, utilizando una media ponderada o el valor más repetido según la distancia.

Predictive Mean Matching

La correspondencia predictiva de medias (*Predictive Mean Matching*) calcula el valor predicho de la variable objetivo según el modelo de imputación especificado. Para cada entrada que falta, el método forma un pequeño conjunto de donantes candidatos (normalmente con 3, 5 o 10 miembros) a partir de todos los casos completos que tienen los valores predichos más cercanos al valor predicho para la entrada que falta. Se extrae aleatoriamente un donante de entre los candidatos y se toma el valor observado del donante para sustituir el valor que falta. Se supone que la distribución de la celda que falta es la misma que la de los datos observados de los donantes candidatos (Van Buuren, 2018).

Es un método fácil de utilizar y versátil. Es bastante robusto a las transformaciones de la variable objetivo. El método también permite variables objetivo discretas. Las imputaciones se basan en valores observados en otros lugares, por lo que son realistas. No se producirán imputaciones fuera del rango de datos observados, evitando así problemas con imputaciones sin sentido (por ejemplo, altura corporal negativa). El modelo es implícito (Little y Rubin 2002), lo que significa que no es necesario definir un modelo explícito para la distribución de los valores que faltan. Debido a esto, el emparejamiento predictivo de medias es menos vulnerable a la especificación errónea del modelo que otros métodos (Van Buuren, 2018).

"*PMM* existe desde hace mucho tiempo (Rubin 1986, Little 1988), pero sólo recientemente se ha generalizado su uso y ha pasado a ser práctico. Originalmente, sólo podía utilizarse en situaciones en las que una única variable tenía datos perdidos o, más ampliamente, cuando el patrón de datos perdidos era monotónico. Ahora, sin embargo, el método *PMM* está integrado en muchos paquetes de software que implementan un enfoque de

imputación múltiple conocido como imputación múltiple por ecuaciones encadenadas (MICE), regresión generalizada secuencial o especificación totalmente condicional (FCS). Está disponible en muchos paquetes estadísticos, incluidos SAS, Stata, SPSS y R, todos los cuales permiten utilizar PMM para prácticamente cualquier patrón de datos perdidos" (Allison, 2015).

Para imputar los datos con este método en R vamos a utilizar la función *mice* indicando *method = "pmm"*, del paquete *mice*.

3.1.2. Clasificación

Una vez imputados los datos mediante las diferentes técnicas seleccionadas, se van a utilizar los datos para realizar la técnica de minería de datos de clasificación, donde a partir de las variables observadas, se obtendrá una predicción de la variable respuesta y se hará una clasificación de las observaciones que han sido predichas correctamente y las que no.

Aprender como clasificar objetos a una de las categorías o clases previamente establecidas, es una característica de la inteligencia de máximo interés para investigadores, dado que la habilidad de realizar una clasificación y de aprender a clasificar otorga el poder de tomar decisiones.

La clasificación es el proceso de dividir un conjunto de datos en grupos mutuamente excluyentes, de tal forma que cada miembro de un grupo esté lo más cerca posible de otros y grupos diferentes estén lo más lejos posible de otros, donde la distancia se mide con respecto a las variables especificadas como objeto de predicción.

Es una técnica importante en la minería de datos que se utiliza para asignar ejemplos a categorías o clases, y existen muchos algoritmos diferentes disponibles para llevar a cabo esta tarea. Es una técnica ampliamente utilizada en una gran variedad de contextos y sigue siendo un tema de investigación activo en el campo de la inteligencia artificial y el análisis de datos.

Clasificación bayesiana

Para este análisis se va a utilizar el clasificador de *naive* bayes. Los clasificadores Bayesianos son clasificadores estadísticos, que pueden predecir tanto las probabilidades del número de miembros de clase, como la probabilidad de que una muestra dada pertenezca a una clase particular. La clasificación Bayesiana se basa en el teorema de Bayes, y los clasificadores Bayesianos han demostrado una alta exactitud y velocidad cuando se han aplicado a grandes bases de datos, como es nuestro caso.

A continuación se explican los fundamentos de los clasificadores bayesianos y, más concretamente, del clasificador *naive* Bayesiano.

Clasificador Naive Bayesiano

Lo que normalmente se quiere saber en aprendizaje es cuál es la mejor hipótesis (más probable) dados los datos. Si denotamos $P(D)$ como la probabilidad a priori de los datos (i.e., que datos son más probables que otros), $P(D|h)$ la probabilidad de los datos dada una hipótesis, lo que queremos estimar es: $P(h|D)$, la probabilidad a posteriori de h dados los datos. La siguiente ecuación muestra cómo se puede estimar usando el teorema de Bayes:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Para estimar la hipótesis más probable (*MAP, maximum a posteriori* hipótesis) se busca el mayor $P(h|D)$ como se muestra en la siguiente ecuación.

$$h_{MAP} = argmax_{h \in H}(P(h|D)) = argmax_{h \in H}\frac{P(D|h)P(h)}{P(D)} = argmax_{h \in H}P(D|h)P(h)$$

Ya que $P(D)$ es una constante independiente de h. Si se asume que todas las hipótesis son igualmente probables, entonces resulta la hipótesis de máxima verosimilitud (*ML, [maximum likelihood]*) de la ecuación:

$$h_{ML} = argmax_{h \in H}P(D|h)$$

El clasificador *naive* Bayesiano se utiliza cuando se quiere clasificar un ejemplo descrito por un conjunto de atributos (a_i's) en un conjunto finito de clases (V). Se pretende clasificar un nuevo ejemplo de acuerdo con el valor más probable dados los valores de sus atributos. Si se aplica la ecuación anterior al problema de la clasificación se obtendrá la siguiente:

$$\begin{aligned} v_{MAP} &= argmax_{v_j \in V}(P(v_j|a_1, ..., a_n)) = argmax_{v_j \in V}\frac{P(a_1, ..., a_n|v_j)P(v_j)}{P(a_1, ..., a_n)} \\ &= argmax_{v_j \in V}P(a_1, ..., a_n|v_j) \end{aligned}$$

Además, el clasificador *naive* Bayesiano asume que los valores de los atributos son condicionalmente independientes dado el valor de la clase, por lo que se hace cierta la ecuación

$$P(a_1, ..., a_n|v_j)P(v_j) = \prod_i P(a_i|v_j)P(v_j)$$

y, con ella:

$$argmax_{v_j \in V} P(v_j|a_1, ..., a_n) = argmax_{v_j \in V} P(v_j) \prod_i P(a_i|v_j)$$

Los clasificadores *naive* Bayesianos asumen que el efecto de un valor del atributo en una clase dada es independiente de los valores de los otros atributos. Esta suposición se llama "independencia condicional de clase". Esta simplifica los cálculos involucrados y, en este sentido, es considerado "ingenuo" de ahí su nombre *naive*. Esta suposición es una simplificación de la realidad. A pesar del nombre del clasificador y de la simplificación realizada, el *naive* Bayesiano funciona muy bien, sobre todo cuando se filtra el conjunto de atributos seleccionado para eliminar redundancia, con lo que se elimina también dependencia entre datos.

Para aplicar esta técnica de minería de datos se va a utilizar la función *naiveBayes* del paquete *e1071* de R.

SVM

La teoría de las Máquinas de Soporte Vectorial (SVM, *Support Vector Machine*) está basada en la idea de la minimización de riego estructural (SRM). En muchas aplicaciones, las SVM han mostrado tener gran desempeño, más que las máquinas de aprendizaje tradicional como las redes neuronales y han sido introducidas como herramientas poderosas para resolcer problemas de clasificación. (Betancourt, 2005)

Las máquinas de vectores soporte (SVM) son una técnica de clasificación basada en márgenes óptimos en el aprendizaje automático. La SVM es un clasificador lineal binario que se ha ampliado a datos no lineales mediante *Kernels* y a datos multiclase mediante diversas técnicas como uno contra uno, uno contra resto, *Crammer Singer SVM*, *Weston Watkins SVM* y grafo acíclico dirigido SVM (DAGSVM), etc. La SVM con un *Kernel* lineal se denomina SVM lineal y la que tiene un *Kernel* no lineal se denomina SVM no lineal. La SVM lineal es una técnica eficiente para aplicaciones de datos de alta dimensión como clasificación de documentos, desambiguación del sentido de la palabra, diseño de fármacos, etc. porque en estas aplicaciones de datos, la precisión de la prueba de la SVM lineal está más cerca de la SVM no lineal, mientras que su entrenamiento es mucho más rápido que el de la SVM no lineal. La SVM está en continua evolución desde sus inicios y los investigadores han propuesto muchas formulaciones de problemas, solucionadores y estrategias para resolver la SVM. Por otra parte, debido a los avances en la tecnología, los datos han tomado la forma de 'Big Data' que han planteado un reto para el aprendizaje automático para entrenar un clasificador en estos datos a gran escala. (Chauhan et al.,

2019)

Para aplicar esta técnica en R se ha utilizado la función *svm* del paquete *e1071*.

Análisis Discriminante

El Análisis Discriminante Lineal o Linear Discrimiant Analysis (LDA) es una técnica estadística multivariante que se utiliza para realizar clasificaciones de individuos en grupos definidos previamente. Estableceremos la relación existente entre el grupo de pertenencia y los valores observados de las variables para que, cuando un nuevo individuo deba ser clasificado, solamente observando las variables de interés podamos clasificarlo en el grupo más probable y podamos calcular esta probabilidad de pertenencia.

Haciendo uso del teorema de Bayes, LDA estima la probabilidad de que una observación, dado un determinado valor de los predictores, pertenezca a cada una de las clases de la variable cualitativa, $P(Y = k|X = x)$. Finalmente se asigna la observación a la clase k para la que la probabilidad predicha sea mayor. Es una alternativa a la regresión logística cuando la variable cualitativa tiene más de dos niveles (Amat, 2016). Si bien existen extensiones de la regresión logística para múltiples clases, el LDA presenta una serie de ventajas:

- Si las clases están bien separadas, los parámetros estimados en el modelo de regresión logística son inestables. El método de LDA no sufre este problema.
- Si el número de observaciones es bajo y la distribución de los predictores es aproximadamente normal en cada una de las clases, LDA es más estable que la regresión logística.

(Amat, 2016)

El proceso de un análisis discriminante puede resumirse en 6 pasos:

1. Disponer de un conjunto de datos de entrenamiento (training data) en el que se conoce a que grupo pertenece cada observación.
2. Calcular las probabilidades previas (prior probabilities): la proporción esperada de observaciones que pertenecen a cada grupo.
3. Determinar si la varianza o matriz de covarianzas es homogénea en todos los grupos. De esto dependerá que se emplee LDA o QDA.
4. Estimar los parámetros necesarios para las funciones de probabilidad condicional, verificando que se cumplen las condiciones para hacerlo.
5. Calcular el resultado de la función discriminante. El resultado de esta determina a qué grupo se asigna cada observación.

6. Utilizar validación cruzada (cross-validation) para estimar las probabilidades de clasificaciones erróneas.

(Amat, 2016)

Las tres hipótesis que deben cumplir los datos para poder realizar este tipo de análisis son: normalidad, homocedasticidad y no multicolinealidad.

Para aplicar esta técnica en R se utilizará la función *lda* del paquete *MASS*.

3.2. Aplicación

Para aplicar lo tratado hasta ahora, se van a utilizar dos conjuntos de datos, *"UCI Nursery Data Set"* y *"UCI Mushroom Data"* que podemos encontrar en la página web de *Kaggle*, accediendo mediante los enlaces `https://www.kaggle.com/datasets/heitornunes/nursery` y `https://www.kaggle.com/code/aavigan/uci-mushroom-data/data`, respectivamente.

3.2.1. Descripción de los datos

UCI Nursery Data Set

La base de datos *Nursery* procede de un modelo de decisión jerárquico desarrollado originalmente para clasificar las solicitudes de ingreso en guarderías. Se utilizó durante varios años en la década de 1980, cuando había un exceso de matriculaciones en estas escuelas en Liubliana (Eslovenia), y las solicitudes rechazadas necesitaban con frecuencia una explicación objetiva. La decisión final dependía de tres subproblemas: ocupación de los padres y guardería del niño, estructura familiar y situación económica, y situación social y sanitaria de la familia. El modelo se desarrolló con el sistema experto para la toma de decisiones DEX (M. Bohanec, V. Rajkovic: Expert system for decision making. Sistemica 1(1), pp. 145-157, 1990).

El objetivo va a ser comparar los algoritmos de clasificación aplicados a estos datos para obtener un modelo que a partir de las variables explicativas sobre la ocupación de los padres, guardería del niño, estructura familiar, situación económica, y situación social y sanitaria de la familia se pueda clasificar cómo de recomendable es la admisión en la guardería. Además, se van a aplicar los métodos de imputación mencionados a datos faltantes, para ver el efecto de este enfoque a la hora de tratar la ausencia de datos, y ver cual de ellos es el más adecuado para este tipo de datos.

Este conjunto de datos contiene 12630 observaciones y 9 variables. La última es la variable objetivo, final.evaluation, que tiene 3 clases: not recommended (no recomendada),

acceptance (acceptación), priority (prioridad). El resto de variables y sus posibles valores se recogen en la siguiente tabla:

Variable	Descripción
parents (ocupación de los padres)	usual (usuales) pretentious (pretenciosos) great_pret (de gran pretensión)
has_nurs (guardería infantil)	proper (adecuado) less_proper (menos adecuado) improper (inapropiado) critical (crítico) very_crit (muy crítico)
form (forma de la familia)	complete (completo) completed (finalizado) incomplete (incompleto) foster (en acogida)
children (nº de hijos)	1 2 3 more (más)
housing (condiciones de vivienda)	convenient (conveniente) less_conv (menos conveniente) critical (crítico)
finance (situación económica de la familia)	convenient (conveniente) inconv (inconveniente)
social (condiciones sociales)	nonprob (no problemático) slightly_prob (ligeramente problemático) problematic (problemático)
health (condiciones de salud)	not_recom (no recomendado) recommended (recomendado) priority (prioridad)
final.evaluation (evaluación final)	not_recom (no recomendado) recommend (recomendado) priority (prioridad)

Los datos originales no tienen ningún dato faltante, por lo tanto, vamos a eliminar algunos datos de forma aleatoria y comparar los resultados obtenidos imputando esos

datos faltantes con los obtenidos con los datos reales y, de esta forma, comprobar como de efectivo resulta este enfoque.

Para este análisis se han transformado todas las variables a factores para que R tenga en cuenta su naturaleza a la hora de tratarlos. En primer lugar se ha decidido de forma aleatoria cuantas observaciones se van a sacar de cada variable, sin superar el 10 %, como mucho se podrían extraer 1200 observaciones de cada variable. En segundo lugar se ha decidido (de forma aleatoria también) qué observaciones se van a extraer de cada una de ellas. El número de observaciones extraídas se puede consultar en el Anexo así como el código íntegro utilizado para todo el capítulo.

UCI Mushroom Data

El conjunto de datos *UCI Mushroom Data*, a partir de ahora mencionado como *Mushroom* contiene las observaciones de 8124 setas y 23 variables que describen sus principales características. Para facilitar el uso de este conjunto de datos se usarán 14 de esas variables, entre las cuales se encuentra la variable objetivo de clasificación cuyos posibles valores son edible (comestible) y poisonous (venenosa).

El objetivo va a ser, de nuevo, comparar varios algoritmos de clasificación aplicados a estos datos para obtener un modelo que, a partir de las variables explicativas (características de las seta), se pueda clasificar la seta como comestible o venenosa. Se clasificarán los datos obtenidos al aplicar varios métodos de imputación.

La descripción de las variables de este conjunto de datos, y los valores que pueden tomar cada una de ellas, se recogen en la siguiente tabla:

Variable	Descripción
class (clase)	e=edible (comestible) p=poisonous (venenosa)
bruises (magulladura)	t=bruises (con magulladuras) f=no bruises (sin magulladuras)

odor (hedor)	a=almond (almendra) l=anise (anís) c=creosote (cresol) y=fishy (a pescado) f=foul (fétido) m=musty (a humedad) n=none (ninguno) p=pungent (intenso) s=spicy (picante)
gill.attachment (sujeción de las láminas)	a=attached (sujeto) d=descending (descendiente) f=freen (libre) n=notched (agujereado)
gill.spacing (espacio de las láminas)	c=close (cercano) w=crowded (abarrotado) d=distant (distante)
gill.size (tamaño de las láminas)	b=broad (ancho) n=narrow (estrecho)
stalk.shape (forma del tallo)	e=enlarging (extenso) t=tapering (estrecho)
stalk.root (raíz del tallo)	b=bulbous (protuberante) c=club () u=cup (copa) e=equal (igual) z=rhizomorphs () r=rooted (arraigada) ?=missing (perdido)
stalk.color.below.ring (color del tallo debajo del anillo)	n=brown (marrón) b=buff (beige) c=cinamon (canela) g=gray (gris) o=orange (naranja) p=pink (rosa) e=red (rojo)

veil.color (color de velo)	n=brown (marrón) o=orange (naranja) w=white (blanco) y=yellow (amarillo)
ring.number (nº de anillos)	n=none (ninguno) o=one (uno) t=two (dos)
ring.type (tipo de anillos)	c=cobwebby (fino) e=evanescent (evanescente) f=flaring (enardecido) l=large (largos) n=none (ninguno) p=pendant (colgantes) s=sheathing (enfundados) z=zone (separados)
spore.print.color (color de las pintas de las esporas)	k=black (negro) n=brown (marrón) b=buff (beige) h=chocolate (chocolate) r=green (verde) o=orange (naranja) u=purple (morado) w=white (blanco) y=yellow (amarillo)
population (población)	a=abundant (abundante) c=clustered (agrupada) n=numerous (numerosa) s=scattered (disperso) v=several (varios) y=solitary (solitaria)

De nuevo, los datos originales no tienen ningún dato faltante, por lo tanto, vamos a eliminar algunos datos de forma aleatoria y comparar los resultados obtenidos imputando esos datos faltantes con los obtenidos con los datos reales y, de esta forma, comprobar como de efectivo resulta este enfoque.

Para este análisis se han transformado todas las variables a factores para que R tenga

en cuenta su naturaleza a la hora de tratarlos. En primer lugar se ha decidido de forma aleatoria cuantas observaciones se van a sacar de cada variable, sin superar el 10 %, como mucho se podrían extraer 800 observaciones de cada variable. En segundo lugar se ha decidido (de forma aleatoria también) qué observaciones se van a extraer de cada una de ellas. De nuevo, el número de observaciones extraídas se puede consultar en el Anexo así como el código íntegro utilizado para todo el capítulo.

3.2.2. Resultados

Imputación

La tasa de error de cada variable del conjunto de datos *Nursery*, obtenida mediante los distintos métodos de imputación anteriormente mencionados, se recoge en la siguiente tabla:

	Tasa de error (%)			
	MC	**KNNI**	**KNNIW**	**PMM**
parents	63.03237	66.09881	66.78024	59.62521
has_nurs	79.64377	81.29771	82.18830	72.39186
form	76.73179	85.79041	87.83304	74.77798
children	76.85353	87.16094	89.33092	74.14105
housing	72.89720	77.57009	81.30841	66.35514
finance	50.00000	60.36585	58.94309	51.82927
social	66.61850	79.62428	80.78035	67.63006
health	66.50367	36.18582	36.91932	34.47433
final.evaluation	62.93103	17.38506	17.38506	10.63218

Tabla 3.3: Comparación de la tasa de error obtenida con los distintos métodos de imputación

En el caso de la imputación mediante el valor global más común del atributo para los atributos simbólicos y valor global medio para los atributos simbólicos para este conjunto de datos, al ser todos los atributos simbólicos, se sustituirán los valores faltantes por el valor más común de la variable, que son:

- parents: great_pret
- has_nurs: very_crit
- form: foster

- children: more
- housing: critical
- finance: inconv
- social: problematic
- health: not_recom
- final.evaluation: not_recom

Una vez imputados los datos, se puede observar en la tabla 3.3 que la imputación no es muy acertada en general, el error cometido está entre 50 % y 79.64 %. Las variables con menor tasa de error de imputación son finance con un 50 % y final.evaluation con un 62.93103 %, mientras que las que mayor tasa de error de imputación tienen son has_nurs con un 79.64377 % y children con un 76.85353 %.

Estos resultados desfavorables podrían ser explicados por varios factores, en primer lugar, las variables has_nurs, form y children son las que tienen un mayor número de datos faltantes (786, 1126 y 1106, respectivamente). Además, son variables que tienen bastantes clases por lo que al imputar por la moda, estas se desbalancean más, y la clase más común amplía su ventaja sobre las demás, como podemos observar en las siguientes tablas, donde se recogen las frecuencias de cada clase antes y despúes de imputar los datos faltantes de dichas variables:

Variable	has_nurs				
Valores	proper	less_proper	improper	critical	very_crit
Datos originales	2460	2460	2526	2592	2592
Datos imputados MC	2312	2312	2364	2424	3218

Variable	form				children			
Valores	complete	completed	incomplete	foster	1	2	3	more
Datos originales	3120	3140	3170	3200	3090	3140	3200	3200

Datos imputados MC	2825	2851	2890	4064	2825	2867	2888	4050

Se ve que la clase más común de has_nurs pasa de 2592 a 3218 observaciones (700 más aproximadamente), la clase más común de form pasa de 3200 a 4064 (800 más aproximadamente), un comportamiento similar tiene la de children también, pasa de 3200 a 4050 observaviones (de nuevo, 800 más aproximadamente). Esto refleja claramente las desventajas de este método de imputación, mencionadas cuando se introdujo por primera vez en este trabajo: subestima la varianza, altera las relaciones entre las variables y sesga casi cualquier estimación que no sea la media .

Si imputamos estos datos mediante *K-Nearest Neighbor* se puede observar que hay variables donde esta imputación es bastante acertada y otras donde el error cometido es muy grande. Las variables con menor tasa de error de imputación son final.evaluation con un 17.38506 % y health con un 36.18582 %, mientras que las que mayor tasa de error de imputación tienen son children con un 87.16094 % y form con un 85.79041 %.

Los resultados obtenidos por *Weighted K-Nearest Neighbor*, son muy parecidos a los obtenidos con el método anterior, es decir, sin ponderación. Las variables con menor tasa de error de imputación son final.evaluation con un 17.38506 % y health con un 36.91932 %, mientras que las que mayor tasa de error de imputación tienen son children con un 89.33092 % y form con un 87.83304 %.

Mediante *PMM* las variables con menor tasa de error de imputación son final.evaluation con un 10.63218 % y health con un 34.47433 %, mientras las variables en donde se comete un mayor error son form y children con un 74.77798 % y 74.14105 %, respectivamente.

Se puede observar que, en general, los resultados obtenidos no son muy buenos y, por tanto, las imputaciones no son muy acertadas.

Teniendo en cuenta esto, a la vista de los datos de la tabla, el mejor método de imputación atendiendo a la tasa de error cometido en cada variable, es *PMM*, con una tasa notablemente menor en casi todas las variables, sobre todo en la variable respuesta final.evaluation, donde tan solo se comete un 10.63218 % de error al imputar los datos.

Si reflejamos ahora los resultados obtenidos para el conjunto de datos *Mushroom*:

	Tasa de error (%)			
	MC	**KNN**	**WKNN**	**PMM**
class	44.642857	0	0	0.3968254
bruises	50.000000	0	0	1.9230769

odor	53.386454	19.1235060	19.1235060	27.4900398
gill.attachment	3.759398	0	0	0
gill.spacing	15.573770	1.6393443	1.6393443	1.6393443
gill.size	28.260870	0	0	0.2898551
stalk.shape	45.336788	0.2590674	0.2590674	1.2953368
stalk.root	51.075269	1.0752688	1.0752688	10.2150538
stalk.color.below.ring	47.633588	40.1526718	36.7938931	45.0381679
veil.color	3.293413	2.0958084	2.0958084	1.7964072
ring.number	8.991228	0	0	0
ring.type	51.322751	0	0	3.9682540
spore.print.color	72.115385	27.8846154	27.8846154	25.9615385
population	49.732620	32.7985740	33.6898396	43.3155080

Tabla 3.6: Comparación de la tasa de error obtenida con los distintos métodos de imputación

En la imputación por el valor global más común del atributo para los atributos simbólicos y valor global medio para los atributos simbólicos, se han sustituído los valores faltantes de cada variable por los siguientes:

- class: e
- bruises: f
- odor: n
- gill.attachment: f
- gill.spacing: c
- gill.size: b
- stalk.shape: t
- stalk.root: b
- stalk.color.below.ring: w
- veil.color: w
- ring.number: o

- ring.type: p
- spore.print.color: w
- population: v

Como se puede ver en la tabla 3.6, el porcentaje de error cometido está entre el 3.293413 % y el 72.115385 %, un rango bastante amplio. Las variables con menor tasa de error de imputación son veil.color con un 3.293413 % y gill.attachment con un 3.759398 %.

Como se comentó con el conjunto de datos *Nursery*, los resultados más desfavorables de la imputación por el valor más común se obtienen en aquellas variables las cuales, además de ser en las que mayor número de datos faltantes encontramos, tienen un alto número de clases de respuesta. En este caso, en la variable stalk.color.below.ring se imputan 655 datos y en la variable ring.type, 756. Ambas tienen un gran número de variable de respuestas, por lo que al imputar estos datos por el valor más común se produce un desbalanceo de las clases más pronunciado. La clase de respuesta "w" de stalk.color.below.ring aumenta aproximadamente 300 observaciones, y la de ring.type, "p", en aproximadamente 400, como se recoge en la tabla siguiente:

Variable	stalk.color.below.ring									ring.type				
Valores	b	c	e	g	n	o	p	w	y	e	f	l	n	p
Datos originales	432	36	96	576	512	192	1872	4384	24	2776	48	1296	36	3968
Datos imputados MC	382	35	91	522	468	178	1733	4696	19	2517	43	1175	33	4356

Por otro lado, la imputación de los datos faltantes mediante *K-Nearest Neighbor* comete un error entre el 0 % y el 40.1526718 %. En seis variables no se comete ningún error de imputación, mientras que las que mayor error obtienen son stalk.color.below.ring con un 40.1526718 % y population con un 32.7985740 %.

Resultados muy parecidos se obtienen con *Weighted K-Nearest Neighbor*, se comete un error entre el 0 % y el 36.7938931 %. De nuevo, en seis variables no se comete ningún error de imputación, mientras que las que mayor error obtienen son stalk.color.below.ring con un 36.7938931 % y population con un 33.6898396 %.

En el caso de *PMM*, el porcentaje de error cometido se encuentra entre el 0 % de la variable ring.number y el 45.0381679 % de la variable stalk.color.below.ring.

En contraposición con los resultados obtenidos con el conjunto de datos *Nursery*, en

este caso, *PMM* no es el mejor método de imputación en cuanto a tasa de error, sino que *KNN* y *WKNN* obtienen resultados mejores, con un 0 % de error en seis variables, lo cual es bastante destacable. Entre estos dos métodos solo hay ligeras diferencias en las variables de stalk.color.below.ring y population.

Además, como se puede observar, se obtienen resultados de imputación mucho más acertados que en el caso del otro conjunto de datos.

Clasificación

Para realizar la clasificación de los datos en ambos casos se ha dividido la muestra en dos, la muestra de entrenamiento (dos tercios del total) y la muestra de test (el tercio restante), de forma aleatoria. Se han construído los modelos (con los distintos clasificadores mencionados) con la muestra de entrenamiento y después se ha predicho la variable respuesta de la muestra de test.

En el conjunto de datos de *Nursery*, la muestra de entrenamiento consta de 8464 observaciones y la muestra de test de 4166.

Los resultados de clasificación mediante los clasificadores mencionados anteriormente, sobre los datos imputados con las distintas técnicas de imputación, se recogen en la siguiente tabla:

		Clasificación	
Clasificador	**Imputación**	**Clasificaciones correctas**	**Exactitud**
Naive Bayes	**NI**	3520	89.45 %
	MC	3524	84.59 %
	KNN	3774	90.59 %
	WKNN	3780	90.73 %
	PMM	3790	90.97 %
	Datos originales	3821	91.72 %
SVM	**NI**	941	39.22 %
	MC	3587	86.10 %
	KNN	3886	93.28 %
	WKNN	3886	93.28 %
	PMM	3904	93.71 %
	Datos originales	3923	94.17 %
Análisis Discriminante	**NI**	1424	56.22 %
	MC	3543	85.05 %

KNN	2308	55.40 %
WKNN	2314	55.54 %
PMM	2317	55.62 %
Datos originales	3797	91.14 %

Tabla 3.8: Comparación del número de clasificaciones correctas y exactitud de cada clasificador y método de imputación.

Mediante el clasificador *Naive* Bayes se ha llegado a una clasificación con una exactitud de 84.59 % en el caso de la realizada con los datos imputados por *MC* (3524 observaciones clasificadas correctamente), de 90.59 % con los datos imputados por *KNN* (3774 observaciones clasificadas correctamente), de 90.73 % con los datos imputados por *WKNN* (3780 observaciones clasificadas correctamente) y de 90.93 % con los datos imputados por *PMM* (3790 observaciones clasificadas correctamente). Como podemos ver, la diferencia entre estas tres últimas es muy pequeña.

Si hacemos esta clasificación con los datos reales, donde no hay ningún dato faltante obtenemos una exactitud de 91.72 % con 3821 observaciones clasificadas corrrectamente. Sin embargo, si se decide no imputar los datos faltantes, las observaciones clasificadas correctamente descienden a un total de 3520, lo que supone una notable diferencia respecto a la clasificación con datos imputados.

La clasificación mediante SVM con los datos imputados por *MC* logra una exactitud de 86.10 % (3587 observaciones correctamente clasificadas), de 93.28 % con los datos imputados por *KNN* y *WKNN* (3886 observaciones clasificadas correctamente) y de 93.71 % con los datos imputados por *PMM* (3923 observaciones clasificadas correctamente). Si se realiza esta clasificación con los datos originales, sin ningún dato faltante que necesite ser imputado, se obtiene una exactitud de 94.17 % con 3923 observaciones clasificadas correctamente. Finalmente, si se decidiera no imputar los datos faltantes se obtendría una exactitud de 39.22 % con 941 observaciones bien clasificadas.

Mediante Análisis Discriminante, la clasificación con los datos imputados por *MC* logra una exactitud de 85.05 % (3543 observaciones correctamente clasificadas), de 55.40 % con los datos imputados por *KNN* (2308 observaciones clasificadas correctamente), de 55.54 % con los datos imputados por *WKNN* (2314 observaciones clasificadas correctamente) y de 55.62 % con los datos imputados por *PMM* (2317 observaciones clasificadas correctamente). Este descenso brusco de la exactitud con algunos datos imputados se debe a que, como nos advierte R al ejecutar el código, las variables son colineales, y, como hemos mencionado, la

hipótesis de no colinealidad es fundamental para poder utilizar el Análisis Discriminante. Si se realiza esta clasificación con los datos originales, sin ningún dato faltante que necesite ser imputado, se obtiene una exactitud de 91.14 % con 3797 observaciones clasificadas correctamente. Con los datos sin imputar se obtiene un 56.22 % con 1424 clasificaciones correctas.

Por otro lado, con el conjunto de datos *Mushroom*, la muestra se divide en 5444 observaciones de entrenamiento y 2680 de test, y los resultados de la clasificación son los siguientes:

		Clasificación	
Clasificador	**Imputación**	**Clasificaciones correctas**	**Exactitud**
Naive Bayes	**NI**	2455	97.23 %
	MC	2520	94.03 %
	KNN	2620	97.76 %
	WKNN	2620	97.76 %
	PMM	2613	97.5 %
	Datos originales	2610	97.39 %
SVM	**NI**	684	54.72 %
	MC	2598	96.94 %
	KNN	2680	100 %
	WKNN	2680	100 %
	PMM	2674	99.78 %
	Datos originales	2680	100 %
Análisis Discriminante	**NI**	1336	100 %
	MC	2605	97.2 %
	KNN	2680	100 %
	WKNN	2680	100 %
	PMM	2663	99.37 %
	Datos originales	2678	99.93 %

Tabla 3.9: Comparación del número de clasificaciones correctas y exactitud de cada clasificador y método de imputación.

La clasificación obtenida mediante *Naive* Bayes consigue una exactitud de 94.03 % en

el caso de la clasificación realizada sobre los datos imputados por *MC* (2520 observaciones clasificadas correctamente), de 97.76 % con los datos imputados por *KNN* (2620 observaciones clasificadas correctamente), de 97.76 % con los datos imputados por *WKNN* (2620 observaciones clasificadas correctamente) y de 97.5 % con los datos imputados por *PMM* (2613 observaciones clasificadas correctamente). Como podemos ver, la diferencia entre *KNN* y *WKNN* es nula, y la de estas con *PMM* es muy pequeña. La exactitud obtenida con los datos reales es de 97.39 %, con 2610 observaciones clasificadas correctamente, mientras que la obtenida con los datos sin imputar es de 97.23 %, con 2455.

Si utilizamos el clasificador de SVM, obtenemos una exactitud de 96.94 % con los datos imputados por *MC* (2598 observaciones clasificadas correctamente), de 100 % con los datos imputados por *KNN* (2680 observaciones clasificadas correctamente), de 100 % con los datos imputados por *WKNN* (2680 observaciones clasificadas correctamente) y de 99.78 % con los datos imputados por *PMM* (2674 observaciones clasificadas correctamente). Si se realiza esta clasificación con los datos originales, sin ningún dato faltante que necesite ser imputado, se obtiene una exactitud de 100 % con 2680 observaciones clasificadas correctamente. Finalmente, si se decidiera no imputar los datos faltantes se obtendría una exactitud de 54.72 % con 684 observaciones bien clasificadas.

Mediante Análisis Discriminante con los datos imputados por *MC* se clasifican correctamente 2605 observaciones, un 97.2 %; con los datos imputados por *KNN* y *WKNN* 2680, el 100 %; y, con los datos imputados por *PMM*, 2663, un 99.37 %. Con los datos originales se obtiene una exactitud de 99.93 %, 2678, y con los datos sin imputar se pierden observaciones por el camino, y de las que quedan, 1336, todas se clasifican correctamente. De nuevo, R avisa de que las variables son colineales, por lo que este tipo de análisis no sería válido al incumplir los datos una de sus hipótesis.

3.2.3. Conclusiones

Una vez descartado el método de Análisis Discriminante por el incumplimiento de las hipóteisis iniciales necesarias para aplicarlo, la decisión entre los otros dos clasificadores, aunque ajustada, es clara. El SVM ofrece buenos resultados para los datos en ambos conjuntos. En el caso de *Nursery*, especialmente para aquellos imputados mediante *PMM*, con una exactitud del 93.71 %, muy cercana a la obtenida con los datos originales del 94,17 %, menor del 0.5 %.

Imputando los datos mediante este método y clasificándolos con máquinas de vectores de soporte se obtiene una exactitud mayor del 90 % y por tanto muy favorable, lo cual indica que se podría clasificar la idoniedad de recomendar la asistencia a la guardería infantil de un determinado niño a partir de carácterísticas familiares, como la ocupación de los padres,

guardería del niño, estructura familiar, situación económica, y situación social y sanitaria de la familia.

En el caso de los datos *Mushroom*, los resultados obtenidos son incluso mejores, utilizando SVM obtenemos un 100 % de exactitud, tanto con los datos imputados por *KNN* como por *WKNN* que, como se observó antes, son los métodos de imputación que menor error cometían.

Además, estos resultados reflejan el beneficio de los métodos de imputación como tratamiento de datos faltantes a la hora de realizar un análisis, ya que se obtienen resultados muy próximos a los reales, mientras que si se prescindiese de esas observaciones o se decidiera no imputar, se obtendrían resultados más desacertados.

Cabe destacar también que, para realizar la imputación, hay que tener siempre en cuenta el tipo de datos que se utilizan. Aquí está la prueba, mientras que el método de imputación *PMM* obtiene mejores resultados que el resto aplicados al conjunto de datos *Nursery*, *KNN* y *WKNN* son más acertados que el resto de métodos aplicados al conjunto de datos *Mushroom*.

Por lo tanto, hay que tener siempre en cuenta el tipo de datos que se están manejando, así como el análisis que se va a realizar o la técnica de minería de datos que se va a aplicar, ya que son factores que afectan a la decisión del método de imputación a utilizar. Este puede no tener un efecto tan adecuado si no se consideran estas cuestiones y se puede estar desaprovechando información, y, el objetivo de la minería de datos, es precisamente lo contrario, sacar el mayor provecho posible de la información que nos ofrecen los datos.

Bibliografía

[1] Acuna, E., y Rodriguez, C. (2004). The treatment of missing values and its effect on classifier accuracy. In *Classification, clustering, and data mining applications* (pp. 639-647). Springer, Berlin, Heidelberg.

[2] Allison, P. (2015). Imputation by predictive mean matching: Promise & peril. *Statistical Horizons*.

[3] Amat Rodrigo, J. (Septiembre de 2016). *Análisis discriminante lineal (LDA) y análisis discriminante cuadrático (QDA)*. cienciadedatos.net https://www.cienciadedatos.net/documentos/28_linear_discriminant_analysis_lda_y_quadratic_discriminant_analysis_qda

[4] Ballesteros, H. F. V., Iñiguez, E. G., y Velasco, S. R. M. (2018). Minería de Datos. *RECIMUNDO: Revista Científica de la Investigación y el Conocimiento, 2*(1), 339-349.

[5] Barnard, J., y Meng, X. L. (1999). Applications of multiple imputation in medical studies: from AIDS to NHANES. *Statistical methods in medical research, 8(1)*, 17-36.

[6] Barocio, E., Korba, P., Sattinger, W., y Segundo Sevilla, F. R. (2019). Online coherency identification and stability condition for large interconnected power systems using an unsupervised data mining technique. I*ET Generation, Transmission & Distribution, 13*(15), 3323-3333.

[7] Batista, G. E., y Monard, M. C. (2003). An analysis of four missing data treatment methods for supervised learning. *Applied artificial intelligence, 17*(5-6), 519-533.

[8] Betancourt, G. A. (2005). Las máquinas de soporte vectorial (SVMs). *Scientia et technica, 1*(27).

[9] Brown, M. L., y Kros, J. F. (2003). Data mining and the impact of missing data. *Industrial Management & Data Systems, 103*(8), 611-621.

[10] Carmona, P.M., Esquivel, F.J., Maldonado, J.A. y Raya, R.(2022). *Introducción a la Minería de datos* [Apuntes de Minería de datos]. Universidad de Granada, Granada.

[11] Chauhan, V. K., Dahiya, K., y Sharma, A. (2019). Problem formulations and solvers in linear SVM: a review. *Artificial Intelligence Review, 52*(2), 803-855.

[12] Dagnino, J. (2014). *Datos faltantes (missing values). Revista chilena de anestesia, 43*, 332-334.

[13] Fan, C., Xiao, F., Li, Z., y Wang, J. (2018). Unsupervised data analytics in mining big building operational data for energy efficiency enhancement: A review. *Energy and Buildings, 159*, 296-308.

[14] Farhangfar, A., Kurgan, L., y Dy, J. (2008). Impact of imputation of missing values on classification error for discrete data. *Pattern Recognition, 41*(12), 3692-3705.

[15] Fayyad, U., Piatetsky-Shapiro, G., y Smyth, P. (1996). The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM, 39*(11), 27-34.

[16] Grzymala-Busse, J. W., y Hu, M. (2001). A comparison of several approaches to missing attribute values in data mining. In *International Conference on Rough Sets and Current Trends in Computing* (pp. 378-385). Springer, Berlin, Heidelberg.

[17] He, H., Graco, W., y Yao, X. (1999). Application of genetic algorithm and k-nearest neighbour method in medical fraud detection. In *Asia-Pacific Conference on Simulated Evolution and Learning* (pp. 74-81). Springer, Berlin, Heidelberg.

[18] He, H., Wang, J., Graco, W., y Hawkins, S. (1997). Application of neural networks to detection of medical fraud. *Expert systems with applications, 13*(4), 329-336.

[19] Hearty, A. P., y Gibney, M. J. (2008). Analysis of meal patterns with the use of supervised data mining techniques—artificial neural networks and decision trees. *The American journal of clinical nutrition, 88*(6), 1632-1642.

[20] Kirlidog, M., y Asuk, C. (2012). A fraud detection approach with data mining in health insurance. *Procedia-Social and Behavioral Sciences, 62*, 989-994.

[21] Kleinke, K. (2017). Multiple imputation under violated distributional assumptions: A systematic evaluation of the assumed robustness of predictive mean matching. *Journal of Educational and Behavioral Statistics, 42*(4), 371-404.

[22] Kotsiantis, S. B., Pierrakeas, C. J., y Pintelas, P. E. (2003). Preventing student dropout in distance learning using machine learning techniques. In *International conference on knowledge-based and intelligent information and engineering systems* (pp. 267-274). Springer, Berlin, Heidelberg.

[23] Kumar, M., Ghani, R., y Mei, Z. S. (2010). Data mining to predict and prevent errors in health insurance claims processing. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 65-74).

[24] Liou, F. M., Tang, Y. C., y Chen, J. Y. (2008). Detecting hospital fraud and claim abuse through diabetic outpatient services. *Health care management science, 11*(4), 353-358.

[25] Little, R. J., y Rubin, D. B. (2019). *Statistical analysis with missing data* (Vol. 793). John Wiley & Sons.

[26] Luengo, J., García, S., y Herrera, F. (2010). A study on the use of imputation methods for experimentation with radial basis function network classifiers handling missing attribute values: The good synergy between rbfns and eventcovering method. *Neural Networks, 23*(3), 406-418.

[27] Luengo, J., García, S., y Herrera, F. (2012). On the choice of the best imputation methods for missing values considering three groups of classification methods. *Knowledge and information systems, 32*(1), 77-108.

[28] Majumder, A., y Nath, I. (2021). A Study on Various Applications of Data Mining and Supervised Learning Techniques in Business Fraud Detection. In *Machine Learning Applications for Accounting Disclosure and Fraud Detection* (pp. 108-125). IGI Global.

[29] Molina Arias, M. (13/08/2013). Análisis de Sensibilidad. *Ciencia sin eso... locura doble*. `http://url-del-blog.com`. Recuperado el 17 de diciembre de 2022.

[30] Orallo, J. H., Quintana, M. J. R., y Ramírez, C. F. (2004). *Introducción a la Minería de Datos*. Pearson Educación.

[31] Ortega, P. A., Figueroa, C. J., y Ruz, G. A. (2006). A Medical Claim Fraud/Abuse Detection System based on Data Mining: A Case Study in Chile. *DMIN, 6,* 26-29.

[32] Peng, L., y Lei, L. (2005). A review of missing data treatment methods. *Intell. Inf. Manag. Syst. Technol, 1*, 412-419.

[33] Pérez López, C., y Santin González, D. (2007). *Minería de datos. Técnicas y herramientas: técnicas y herramientas*. Editorial Paraninfo.

[34] Rahim, R., Santoso, J. T., Jumini, S., Bhawika, G. W., Susilo, D., y Wibowo, D. (2021). Unsupervised data mining technique for clustering library in Indonesia. *Library Philosophy and Practice (e-journal), 4866*.

[35] Shin, H., Park, H., Lee, J., y Jhee, W. C. (2012). A scoring model to detect abusive billing patterns in health insurance claims. *Expert Systems with Applications, 39*(8), 7441-7450.

[36] Song, Q., y Shepperd, M. (2007). A new imputation method for small software project data sets. *Journal of Systems and Software, 80*(1), 51-62.

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

[38] Tanner, T., y Toivonen, H. (2010). Predicting and preventing student failure-using the k-nearest neighbour method to predict student performance in an online course environment. *International Journal of Learning Technology*.

[39] Tomasevic, N., Gvozdenovic, N., y Vranes, S. (2020). An overview and comparison of supervised data mining techniques for student exam performance prediction. *Computers y education, 143*, 103676.

[40] Van Buuren, S. (2018). Flexible imputation of missing data. CRC press.

[41] Williams, G. (2011). *Data mining with Rattle and R: The art of excavating data for knowledge discovery*. Springer Science & Business Media.

Anexo

```
datosnursery<-read.csv("nursery.csv")
head(datosnursery)
```

```
##   parents has_nurs     form children    housing    finance        social
## 1   usual   proper complete        1 convenient convenient       nonprob
## 2   usual   proper complete        1 convenient convenient       nonprob
## 3   usual   proper complete        1 convenient convenient       nonprob
## 4   usual   proper complete        1 convenient convenient slightly_prob
## 5   usual   proper complete        1 convenient convenient slightly_prob
## 6   usual   proper complete        1 convenient convenient slightly_prob
##        health final.evaluation
## 1 recommended        recommend
## 2    priority         priority
## 3   not_recom        not_recom
## 4 recommended        recommend
## 5    priority         priority
## 6   not_recom        not_recom
```

```
dim(datosnursery)
```

```
## [1] 12960     9
```

```
table(datosnursery$final.evaluation)
```

```
##
##  not_recom   priority  recommend spec_prior very_recom
##       4320       4266          2       4044        328
```

```
indicesout<-which(datosnursery$final.evaluation=="recommend")
indicesout2<-which(datosnursery$final.evaluation=="very_recom")
datosnur<-datosnursery[-c(indicesout,indicesout2),]
```

```
dim(datosnur)
```

```
## [1] 12630     9
```

```
datos<-datosnur
```

```
datos$parents<-factor(datos$parents, levels = c("usual","pretentious", "great_pret"))
datos$has_nurs<-factor(datos$has_nurs, levels = c("proper", "less_proper", "improper",
                                                  "critical", "very_crit"))
datos$form<-factor(datos$form, levels=c("complete", "completed", "incomplete", "foster"))
datos$children<-factor(datos$children, levels=c("1","2","3","more"))
datos$housing<-factor(datos$housing, levels=c("convenient","less_conv", "critical"))
datos$finance<-factor(datos$finance, levels=c("convenient", "inconv"))
datos$social<-factor(datos$social, levels=c("nonprob","slightly_prob", "problematic"))
datos$health<-factor(datos$health, levels=c("not_recom","recommended","priority"))
datos$final.evaluation<-factor(datos$final.evaluation, levels=c("not_recom","priority",
                        "spec_prior"), labels=c("not_recom","recommend", "priority"))
```

```
datosorig<-datos
```

```
table(datos$parents)
```

```
##
##       usual pretentious  great_pret
##        4122        4188        4320
```

```
table(datos$has_nurs)
```

```
##
```

```
##      proper less_proper    improper    critical   very_crit
##        2460        2460        2526        2592        2592
```

```
table(datos$form)
```

```
##
##   complete  completed incomplete     foster
##       3120       3140       3170       3200
```

```
table(datos$children)
```

```
##
##    1    2    3 more
## 3090 3140 3200 3200
```

```
table(datos$housing)
```

```
##
## convenient  less_conv   critical
##       4110       4220       4300
```

```
table(datos$finance)
```

```
##
## convenient     inconv
##       6260       6370
```

```
table(datos$social)
```

```
##
##       nonprob slightly_prob   problematic
##          4155          4155          4320
```

```
table(datos$health)
```

```
##
##   not_recom recommended    priority
##        4320        3990        4320
```

```
table(datos$final.evaluation)
```

```
##
## not_recom recommend  priority
##      4320      4266      4044
```

Se introducen datos faltantes:

```
set.seed(4)
indc<-sample(1:1200,1)
indc
```

```
## [1] 587
```

```
ind<-sample(1:12630,indc)
indc2<-sample(1:1200,1)
ind2<-sample(1:12630,indc2)
indc3<-sample(1:1200,1)
ind3<-sample(1:12630,indc3)
indc4<-sample(1:1200,1)
ind4<-sample(1:12630,indc4)
indc5<-sample(1:1200,1)
ind5<-sample(1:12630,indc5)
indc6<-sample(1:1200,1)
ind6<-sample(1:12630,indc6)
indc7<-sample(1:1200,1)
ind7<-sample(1:12630,indc7)
```

```
indc8<-sample(1:1200,1)
ind8<-sample(1:12630,indc8)
indc9<-sample(1:1200,1)
ind9<-sample(1:12630,indc9)

datos[ind,1]=NA
datos[ind2,2]=NA
datos[ind3,3]=NA
datos[ind4,4]=NA
datos[ind5,5]=NA
datos[ind6,6]=NA
datos[ind7,7]=NA
datos[ind8,8]=NA
datos[ind9,9]=NA

head(datos)
```

```
##   parents has_nurs     form children    housing    finance        social
## 2   usual   proper complete     <NA> convenient convenient       nonprob
## 3   usual   proper complete     <NA> convenient convenient       nonprob
## 5   usual   proper complete        1 convenient convenient slightly_prob
## 6   usual   proper complete        1 convenient convenient slightly_prob
## 7   usual   proper complete        1 convenient convenient          <NA>
## 8   usual   proper complete        1 convenient convenient   problematic
##        health final.evaluation
## 2    priority             <NA>
## 3        <NA>        not_recom
## 5    priority             <NA>
## 6   not_recom        not_recom
## 7 recommended        recommend
## 8    priority        recommend
```

```
datos_mis<-datos
faltantes<-colSums(is.na(datos)) #Para ver cuantos valores faltantes tiene cada columna
faltantes
```

```
##          parents         has_nurs             form         children
##              587              786             1126             1106
##          housing          finance           social           health
##              107              492              692              409
## final.evaluation
##              696
```

```
library(VIM)
```

```
## Warning: package 'VIM' was built under R version 4.1.2
```

```
## Loading required package: colorspace
```

```
## Loading required package: grid
```

```
## VIM is ready to use.
```

```
## Suggestions and bug-reports can be submitted at: https://github.com/statistikat/VIM/issues
```

```
##
## Attaching package: 'VIM'
```

```
## The following object is masked from 'package:datasets':
##
##     sleep
```

```
summary(datos)
```

```
##         parents            has_nurs            form      children
##  usual      :3932   proper     :2312   complete  :2825   1   :2825
##  pretentious:4008   less_proper:2312   completed :2851   2   :2867
##  great_pret :4103   improper   :2364   incomplete:2890   3   :2888
##  NA's       : 587   critical   :2424   foster    :2938   more:2944
##                     very_crit  :2432   NA's      :1126   NA's:1106
##                     NA's       : 786
##        housing           finance               social            health
##  convenient:4070   convenient:6014   nonprob      :3938   not_recom  :4183
##  less_conv :4182   inconv    :6124   slightly_prob:3911   recommended:3862
##  critical  :4271   NA's      : 492   problematic  :4089   priority   :4176
##  NA's      : 107                     NA's         : 692   NA's       : 409
##
##
##   final.evaluation
##  not_recom:4062
##  recommend:4045
##  priority :3827
##  NA's     : 696
##
##
```

Imputación

Imputación MC

```
datos_MC<-datos

getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}


imput<-function(df) {
  for (i in c(1,2,3,4,5,6,7,8,9)){
      index<-which(is.na(df[,i]))
      df[index,i]<-getmode(df[,i])
  }
  return(df)
}

datos_MC<-imput(datos_MC)
```

Se ve que efectivamente que se han imputado los datos y ya no hay ningún dato faltante:

```
colSums(is.na(datos_MC))
```

```
##          parents         has_nurs             form         children
##                0                0                0                0
##          housing          finance           social           health
##                0                0                0                0
## final.evaluation
##                0
```

Cabecera de los datos una vez imputados los faltantes:

```
head(datos_MC)
```

```
##   parents has_nurs     form children    housing    finance        social
## 2   usual   proper complete     more convenient convenient       nonprob
## 3   usual   proper complete     more convenient convenient       nonprob
## 5   usual   proper complete        1 convenient convenient slightly_prob
## 6   usual   proper complete        1 convenient convenient slightly_prob
## 7   usual   proper complete        1 convenient convenient   problematic
## 8   usual   proper complete        1 convenient convenient   problematic
##       health final.evaluation
## 2    priority        not_recom
## 3   not_recom        not_recom
## 5    priority        not_recom
## 6   not_recom        not_recom
## 7 recommended        recommend
## 8    priority        recommend
```

Datos que se han imputado mal:

```
12630-colSums(datos_MC[,1:9]==datosorig)
```

```
##          parents         has_nurs             form         children
##              370              626              864              850
##          housing          finance           social           health
##               78              246              461              272
## final.evaluation
##              438
```

Porcentaje de error:

```
error_MC<-(12630-colSums(datos_MC[,1:9]==datosorig))/faltantes*100
error_MC
```

```
##          parents         has_nurs             form         children
##         63.03237         79.64377         76.73179         76.85353
##          housing          finance           social           health
##         72.89720         50.00000         66.61850         66.50367
## final.evaluation
##         62.93103
```

```
table(datos_MC$parents)
```

```
##
##       usual pretentious  great_pret
##        3932        4008        4690
```

```
table(datos_MC$has_nurs)
```

```
##
##      proper less_proper    improper    critical   very_crit
##        2312        2312        2364        2424        3218
```

```
table(datos_MC$form)
```

```
##
##   complete  completed incomplete     foster
##       2825       2851       2890       4064
```

```
table(datos_MC$children)
```

```
##
##    1    2    3 more
## 2825 2867 2888 4050
```

```
table(datos_MC$housing)
```

```
##
## convenient  less_conv   critical
##       4070       4182       4378
```

```
table(datos_MC$finance)
```

```
##
## convenient     inconv
##       6014       6616
```

```
table(datos_MC$social)
```

```
##
##       nonprob slightly_prob   problematic
##          3938          3911          4781
```

```
table(datos_MC$health)
```

```
##
##   not_recom recommended    priority
##        4592        3862        4176
```

```
table(datos_MC$final.evaluation)
```

```
##
## not_recom recommend  priority
##      4758      4045      3827
```

Imputación KNN:

```
library(VIM)
datos_KNN <-  kNN(datos, variable = c("parents", "has_nurs", "form", "children",
              "housing", "finance", "social", "health", "final.evaluation"), k = 5)
```

Se ve que efectivamente que se han imputado los datos y ya no hay ningún dato faltante:

```
colSums(is.na(datos_KNN))
```

```
##              parents             has_nurs                 form
##                    0                    0                    0
##             children              housing              finance
##                    0                    0                    0
##               social               health     final.evaluation
##                    0                    0                    0
##          parents_imp         has_nurs_imp             form_imp
##                    0                    0                    0
##         children_imp          housing_imp          finance_imp
##                    0                    0                    0
##           social_imp           health_imp final.evaluation_imp
##                    0                    0                    0
```

Cabecera de los datos una vez imputados los faltantes:

```
head(datos_KNN)
```

```
##   parents has_nurs     form children    housing    finance        social
## 1   usual   proper complete        3 convenient convenient       nonprob
## 2   usual   proper complete        2 convenient convenient       nonprob
## 3   usual   proper complete        1 convenient convenient slightly_prob
## 4   usual   proper complete        1 convenient convenient slightly_prob
## 5   usual   proper complete        1 convenient convenient   problematic
```

```
## 6   usual   proper complete        1 convenient convenient   problematic
##        health final.evaluation parents_imp has_nurs_imp form_imp children_imp
## 1    priority        recommend       FALSE        FALSE    FALSE         TRUE
## 2   not_recom        not_recom       FALSE        FALSE    FALSE         TRUE
## 3    priority        recommend       FALSE        FALSE    FALSE        FALSE
## 4   not_recom        not_recom       FALSE        FALSE    FALSE        FALSE
## 5 recommended        recommend       FALSE        FALSE    FALSE        FALSE
## 6    priority        recommend       FALSE        FALSE    FALSE        FALSE
##   housing_imp finance_imp social_imp health_imp final.evaluation_imp
## 1       FALSE       FALSE      FALSE      FALSE                 TRUE
## 2       FALSE       FALSE      FALSE       TRUE                FALSE
## 3       FALSE       FALSE      FALSE      FALSE                 TRUE
## 4       FALSE       FALSE      FALSE      FALSE                FALSE
## 5       FALSE       FALSE       TRUE      FALSE                FALSE
## 6       FALSE       FALSE      FALSE      FALSE                FALSE
```

Datos que se han imputado mal:

```
12630-colSums(datos_KNN[,1:9]==datosorig)
```

```
##          parents         has_nurs             form         children
##              388              639              966              964
##          housing          finance           social           health
##               83              297              551              148
## final.evaluation
##              121
```

Porcentaje de error:

```
error_KNN<-(12630-colSums(datos_KNN[,1:9]==datosorig))/faltantes*100
error_KNN
```

```
##          parents         has_nurs             form         children
##         66.09881         81.29771         85.79041         87.16094
##          housing          finance           social           health
##         77.57009         60.36585         79.62428         36.18582
## final.evaluation
##         17.38506
```

Imputación KNN Weighted:

```
datos_KNN_weighted <-  kNN(datos, variable = c("parents", "has_nurs", "form",
                      "children", "housing", "finance", "social", "health",
                      "final.evaluation"), k = 5, weightDist=TRUE)
```

Se ve que efectivamente que se han imputado los datos y ya no hay ningún dato faltante:

```
colSums(is.na(datos_KNN_weighted))
```

```
##              parents             has_nurs                 form
##                    0                    0                    0
##             children              housing              finance
##                    0                    0                    0
##               social               health     final.evaluation
##                    0                    0                    0
##          parents_imp         has_nurs_imp             form_imp
##                    0                    0                    0
##         children_imp          housing_imp          finance_imp
##                    0                    0                    0
##           social_imp           health_imp final.evaluation_imp
##                    0                    0                    0
```

Porcentaje de error:

```
error_KNN_weighted<-(12630-colSums(datos_KNN_weighted[,1:9]==datosorig))/faltantes*100
error_KNN_weighted
```

```
##          parents         has_nurs             form         children
##         66.78024         82.18830         87.83304         89.33092
##          housing          finance           social           health
##         81.30841         58.94309         80.78035         36.91932
## final.evaluation
##         17.38506
```

Imputación PMM:

```
library(mice)
```

```
## Warning: package 'mice' was built under R version 4.1.2
```

```
##
## Attaching package: 'mice'
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```
imp_multi <- mice(datos, m = 5, method = "pmm")  # Impute missing values multiple times
```

```
##
##  iter imp variable
##   1   1  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   1   2  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   1   3  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   1   4  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   1   5  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   2   1  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   2   2  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   2   3  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   2   4  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   2   5  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   3   1  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   3   2  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   3   3  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   3   4  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   3   5  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   4   1  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   4   2  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   4   3  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   4   4  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   4   5  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   5   1  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   5   2  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   5   3  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   5   4  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
##   5   5  parents  has_nurs  form  children  housing  finance  social  health  final.evaluation
```

```
## Warning: Number of logged events: 105
```

```
datos_PMM<-mice::complete(imp_multi)
```

```
colSums(is.na(datos_PMM))
```

```
##          parents         has_nurs             form         children
##                0                0                0                0
##          housing          finance           social           health
##                0                0                0                0
## final.evaluation
##                0
```

```
error_PMM<-(12630-colSums(datos_PMM[,1:9]==datosorig))/faltantes*100
error_PMM
```

```
##          parents         has_nurs             form         children
##         59.62521         72.39186         74.77798         74.14105
##          housing          finance           social           health
##         66.35514         51.82927         67.63006         34.47433
## final.evaluation
##         10.63218
```

Clasificación

Naive Bayes

Con los **datos imputados por MC**:

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.1.2
```

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.1.2
```

```
## Loading required package: lattice
```

```
set.seed(4)
training_ids<-createDataPartition(datosnur$final.evaluation, p=0.67, list=F)
```

```
model_MC<-naiveBayes(final.evaluation~.,datos_MC[training_ids,])
pred_MC <- predict(model_MC, datos_MC[-training_ids,])
tab_MC<-table(datos_MC[-training_ids,]$final.evaluation, pred_MC,
              dnn=c("Actual","Predicha"))
confusionMatrix(tab_MC)
```

```
## Confusion Matrix and Statistics
##
##            Predicha
## Actual      not_recom recommend priority
##   not_recom      1432        73       62
##   recommend        39      1095      195
##   priority         42       231      997
##
## Overall Statistics
##
##                Accuracy : 0.8459
##                  95% CI : (0.8346, 0.8567)
##     No Information Rate : 0.3632
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                   Kappa : 0.7681
##
##  Mcnemar's Test P-Value : 0.0006399
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                    0.9465           0.7827          0.7951
## Specificity                    0.9491           0.9154          0.9062
## Pos Pred Value                 0.9138           0.8239          0.7850
## Neg Pred Value                 0.9688           0.8928          0.9113
## Prevalence                     0.3632           0.3358          0.3010
## Detection Rate                 0.3437           0.2628          0.2393
## Detection Prevalence           0.3761           0.3190          0.3048
## Balanced Accuracy              0.9478           0.8491          0.8507
```

Con los **datos imputados por KNN**:

```
model_KNN<-naiveBayes(final.evaluation~.,datos_KNN[training_ids,])
pred_KNN<- predict(model_KNN, datos_KNN[-training_ids,])
tab_KNN<-table(datos_KNN[-training_ids,]$final.evaluation, pred_KNN,
               dnn=c("Actual","Predicha"))
confusionMatrix(tab_KNN)
```

```
## Confusion Matrix and Statistics
##
##            Predicha
## Actual      not_recom recommend priority
##   not_recom      1425         0        0
##   recommend         0      1271      165
##   priority          0       227     1078
##
## Overall Statistics
##
##                Accuracy : 0.9059
##                  95% CI : (0.8966, 0.9146)
##     No Information Rate : 0.3596
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8586
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                    1.0000           0.8485          0.8673
## Specificity                    1.0000           0.9382          0.9223
## Pos Pred Value                 1.0000           0.8851          0.8261
## Neg Pred Value                 1.0000           0.9168          0.9423
## Prevalence                     0.3421           0.3596          0.2984
## Detection Rate                 0.3421           0.3051          0.2588
## Detection Prevalence           0.3421           0.3447          0.3133
## Balanced Accuracy              1.0000           0.8933          0.8948
```

Con los **datos imputados por KNN WEIGHTED**:

```
model_KNN_weighted<-naiveBayes(final.evaluation~.,datos_KNN_weighted[training_ids,])
pred_KNN_weighted <- predict(model_KNN_weighted, datos_KNN_weighted[-training_ids,])
tab_KNN_weighted<-table(datos_KNN_weighted[-training_ids,]$final.evaluation,
```

```
                        pred_KNN_weighted, dnn=c("Actual","Predicha"))
confusionMatrix(tab_KNN_weighted)
```

```
## Confusion Matrix and Statistics
##
##            Predicha
## Actual      not_recom recommend priority
##   not_recom      1427         0        0
##   recommend         0      1268      166
##   priority          0       220     1085
##
## Overall Statistics
##
##                Accuracy : 0.9073
##                  95% CI : (0.8981, 0.916)
##     No Information Rate : 0.3572
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8608
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                    1.0000           0.8522          0.8673
## Specificity                    1.0000           0.9380          0.9245
## Pos Pred Value                 1.0000           0.8842          0.8314
## Neg Pred Value                 1.0000           0.9195          0.9420
## Prevalence                     0.3425           0.3572          0.3003
## Detection Rate                 0.3425           0.3044          0.2604
## Detection Prevalence           0.3425           0.3442          0.3133
## Balanced Accuracy              1.0000           0.8951          0.8959
```

Con los **datos imputados por PMM**:

```
model_PMM<-naiveBayes(final.evaluation~.,datos_PMM[training_ids,])
pred_PMM <- predict(model_PMM, datos_PMM[-training_ids,])
tab_PMM<-table(datos_PMM[-training_ids,]$final.evaluation, pred_PMM,
               dnn=c("Actual","Predicha"))
confusionMatrix(tab_PMM)
```

```
## Confusion Matrix and Statistics
##
##            Predicha
## Actual      not_recom recommend priority
##   not_recom      1427         0        0
##   recommend         0      1244      159
##   priority          0       217     1119
##
## Overall Statistics
##
##                Accuracy : 0.9097
##                  95% CI : (0.9006, 0.9183)
##     No Information Rate : 0.3507
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8645
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                    1.0000           0.8515          0.8756
## Specificity                    1.0000           0.9412          0.9249
## Pos Pred Value                 1.0000           0.8867          0.8376
## Neg Pred Value                 1.0000           0.9215          0.9438
## Prevalence                     0.3425           0.3507          0.3068
## Detection Rate                 0.3425           0.2986          0.2686
## Detection Prevalence           0.3425           0.3368          0.3207
## Balanced Accuracy              1.0000           0.8963          0.9002
```

Con los **datos originales**:

```
model_orig<-naiveBayes(final.evaluation~.,datosorig[training_ids,])
pred_orig<- predict(model_orig, datosorig[-training_ids,])
tab_orig<-table(datosorig[-training_ids,]$final.evaluation, pred_orig,
                dnn=c("Actual","Predicha"))
confusionMatrix(tab_orig)
```

```
## Confusion Matrix and Statistics
##
##            Predicha
## Actual      not_recom recommend priority
##   not_recom      1425         0        0
##   recommend         0      1261      146
##   priority          0       199     1135
##
## Overall Statistics
##
##                Accuracy : 0.9172
##                  95% CI : (0.9084, 0.9254)
##     No Information Rate : 0.3505
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8757
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                    1.0000           0.8637          0.8860
## Specificity                    1.0000           0.9460          0.9310
## Pos Pred Value                 1.0000           0.8962          0.8508
## Neg Pred Value                 1.0000           0.9279          0.9484
## Prevalence                     0.3421           0.3505          0.3075
## Detection Rate                 0.3421           0.3027          0.2724
## Detection Prevalence           0.3421           0.3377          0.3202
## Balanced Accuracy              1.0000           0.9049          0.9085
```

Con los **datos sin imputar**:

```
model_mis<-naiveBayes(final.evaluation~.,datos_mis[training_ids,])
pred_mis <- predict(model_mis, datos_mis[-training_ids,])
tab_mis<-table(datos_mis[-training_ids,]$final.evaluation, pred_mis,
               dnn=c("Actual","Predicha"))
confusionMatrix(tab_mis)
```

```
## Confusion Matrix and Statistics
##
##            Predicha
## Actual      not_recom recommend priority
##   not_recom      1303        20       13
##   recommend         4      1170      155
##   priority          7       216     1047
##
## Overall Statistics
##
##                Accuracy : 0.8945
##                  95% CI : (0.8845, 0.904)
##     No Information Rate : 0.3573
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8417
##
##  Mcnemar's Test P-Value : 5.142e-05
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                    0.9916           0.8321          0.8617
## Specificity                    0.9874           0.9371          0.9180
## Pos Pred Value                 0.9753           0.8804          0.8244
## Neg Pred Value                 0.9958           0.9094          0.9370
## Prevalence                     0.3339           0.3573          0.3088
## Detection Rate                 0.3311           0.2973          0.2661
## Detection Prevalence           0.3395           0.3377          0.3227
## Balanced Accuracy              0.9895           0.8846          0.8899
```

SVM

Con los **datos imputados por MC**:

```
library(e1071)
set.seed(4)
modelo_svm_MC <- svm(final.evaluation ~., datos_MC[training_ids,], kernel = "linear",
                     cost = 10, scale = FALSE)
pred_svm_MC<-predict(modelo_svm_MC,datos_MC[-training_ids,])
tab_svm_MC<-table(datos_MC[-training_ids,]$final.evaluation, pred_svm_MC,
                  dnn=c("Actual","Predicha"))
confusionMatrix(tab_svm_MC)
```

```
## Confusion Matrix and Statistics
##
##            Predicha
## Actual      not_recom recommend priority
##   not_recom      1432        71       64
##   recommend        39      1099      191
##   priority         42       172     1056
##
## Overall Statistics
##
##                Accuracy : 0.861
##                  95% CI : (0.8501, 0.8714)
##     No Information Rate : 0.3632
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                   Kappa : 0.7909
##
##  Mcnemar's Test P-Value : 0.001932
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                    0.9465           0.8189          0.8055
## Specificity                    0.9491           0.9186          0.9250
## Pos Pred Value                 0.9138           0.8269          0.8315
## Neg Pred Value                 0.9688           0.9143          0.9119
## Prevalence                     0.3632           0.3221          0.3147
## Detection Rate                 0.3437           0.2638          0.2535
## Detection Prevalence           0.3761           0.3190          0.3048
## Balanced Accuracy              0.9478           0.8687          0.8653
```

Con los **datos imputados por KNN**:

```
modelo_svm_KNN <- svm(final.evaluation ~., datos_KNN[training_ids,], kernel = "linear",
                      cost = 10, scale = FALSE)
pred_svm_KNN<-predict(modelo_svm_KNN,datos_KNN[-training_ids,])
tab_svm_KNN<-table(datos_KNN[-training_ids,]$final.evaluation, pred_svm_KNN, dnn=c("Actual","Predich
confusionMatrix(tab_svm_KNN)
```

```
## Confusion Matrix and Statistics
##
##            Predicha
## Actual      not_recom recommend priority
##   not_recom      1425         0        0
##   recommend         0      1291      145
##   priority          0       135     1170
##
## Overall Statistics
##
##                Accuracy : 0.9328
##                  95% CI : (0.9248, 0.9402)
##     No Information Rate : 0.3423
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8991
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                    1.0000           0.9053          0.8897
## Specificity                    1.0000           0.9471          0.9526
## Pos Pred Value                 1.0000           0.8990          0.8966
## Neg Pred Value                 1.0000           0.9505          0.9493
## Prevalence                     0.3421           0.3423          0.3157
## Detection Rate                 0.3421           0.3099          0.2808
## Detection Prevalence           0.3421           0.3447          0.3133
## Balanced Accuracy              1.0000           0.9262          0.9212
```

Con los **datos imputados por KNN Weighted**:

```
modelo_svm_KNN_weighted <- svm(final.evaluation ~., datos_KNN_weighted[training_ids,],
                               kernel = "linear", cost = 10, scale = FALSE)
pred_svm_KNN_weighted<-predict(modelo_svm_KNN_weighted,datos_KNN_weighted[-training_ids,])
```

```
tab_svm_KNN_weighted<-table(datos_KNN_weighted[-training_ids,]$final.evaluation,
                            pred_svm_KNN_weighted, dnn=c("Actual","Predicha"))
confusionMatrix(tab_svm_KNN_weighted)
```

```
## Confusion Matrix and Statistics
##
##            Predicha
## Actual      not_recom recommend priority
##   not_recom      1427         0        0
##   recommend         0      1291      143
##   priority          0       137     1168
##
## Overall Statistics
##
##                Accuracy : 0.9328
##                  95% CI : (0.9248, 0.9402)
##     No Information Rate : 0.3428
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8991
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                    1.0000           0.9041          0.8909
## Specificity                    1.0000           0.9478          0.9520
## Pos Pred Value                 1.0000           0.9003          0.8950
## Neg Pred Value                 1.0000           0.9499          0.9500
## Prevalence                     0.3425           0.3428          0.3147
## Detection Rate                 0.3425           0.3099          0.2804
## Detection Prevalence           0.3425           0.3442          0.3133
## Balanced Accuracy              1.0000           0.9259          0.9215
```

Con los **datos imputados por PMM**:

```
modelo_svm_PMM <- svm(final.evaluation ~., datos_PMM[training_ids,], kernel = "linear",
                      cost = 10, scale = FALSE)
pred_svm_PMM<-predict(modelo_svm_PMM,datos_PMM[-training_ids,])
tab_svm_PMM<-table(datos_PMM[-training_ids,]$final.evaluation, pred_svm_PMM, dnn=c("Actual","Predich
confusionMatrix(tab_svm_PMM)
```

```
## Confusion Matrix and Statistics
##
##            Predicha
## Actual      not_recom recommend priority
##   not_recom      1427         0        0
##   recommend         0      1265      138
##   priority          0       124     1212
##
## Overall Statistics
##
##                Accuracy : 0.9371
##                  95% CI : (0.9293, 0.9443)
##     No Information Rate : 0.3425
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9056
```

```
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                    1.0000           0.9107          0.8978
## Specificity                    1.0000           0.9503          0.9560
## Pos Pred Value                 1.0000           0.9016          0.9072
## Neg Pred Value                 1.0000           0.9551          0.9512
## Prevalence                     0.3425           0.3334          0.3241
## Detection Rate                 0.3425           0.3036          0.2909
## Detection Prevalence           0.3425           0.3368          0.3207
## Balanced Accuracy              1.0000           0.9305          0.9269
```

Con los **datos originales**:

```
modelo_svm_orig <- svm(final.evaluation~., datosorig[training_ids,], kernel = "linear",
                       cost = 10, scale = FALSE)
pred_svm_orig<-predict(modelo_svm_orig,datosorig[-training_ids,])
tab_svm_orig<-table(datosorig[-training_ids,]$final.evaluation, pred_svm_orig, dnn=c("Actual","Predi
confusionMatrix(tab_svm_orig)
```

```
## Confusion Matrix and Statistics
##
##            Predicha
## Actual      not_recom recommend priority
##   not_recom      1425         0        0
##   recommend         0      1281      126
##   priority          0       117     1217
##
## Overall Statistics
##
##                Accuracy : 0.9417
##                  95% CI : (0.9341, 0.9486)
##     No Information Rate : 0.3421
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9125
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                    1.0000           0.9163          0.9062
## Specificity                    1.0000           0.9545          0.9586
## Pos Pred Value                 1.0000           0.9104          0.9123
## Neg Pred Value                 1.0000           0.9576          0.9555
## Prevalence                     0.3421           0.3356          0.3224
## Detection Rate                 0.3421           0.3075          0.2921
## Detection Prevalence           0.3421           0.3377          0.3202
## Balanced Accuracy              1.0000           0.9354          0.9324
```

Con los **datos sin imputar**:

```
modelo_svm_mis<- svm(final.evaluation ~., datos_mis[training_ids,], kernel = "linear",
                     cost = 10, scale = FALSE)
pred_svm_mis<-predict(modelo_svm_mis,datos_mis[-training_ids,])
indices_not_recom<-which(pred_svm_mis=="not_recom")
```

```
indices_recommend<-which(pred_svm_mis=="recommend")
indices_priority<-which(pred_svm_mis=="priority")
indicespred<-c(indices_not_recom,indices_recommend,indices_priority)
tab_svm_mis<-table(datos_mis[-training_ids,]$final.evaluation[indicespred]==pred_svm_mis)
tab_svm_mis
```

```
##
## FALSE  TRUE
##  1458   941
```

```
#tab_svm_mis<-table(datos_mis[-training_ids,]$final.evaluation, pred_svm_mis, dnn=c("Actual","Predic
#confusionMatrix(tab_svm_mis)
```

Análisis Discriminante

Con los **datos imputados por MC**:

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 4.1.2
```

```
modelo_lda_MC <- lda(final.evaluation ~., data = datos_MC[training_ids,])
pred_lda_MC <- predict(modelo_lda_MC, datos_MC[-training_ids,])
tab_lda_MC<-table(datos_MC[-training_ids,]$final.evaluation, pred_lda_MC$class,
                  dnn = c("Clase real", "Clase predicha"))
confusionMatrix(tab_lda_MC)
```

```
## Confusion Matrix and Statistics
##
##            Clase predicha
## Clase real  not_recom recommend priority
##   not_recom      1432        76       59
##   recommend        39      1100      190
##   priority         42       217     1011
##
## Overall Statistics
##
##                Accuracy : 0.8505
##                  95% CI : (0.8393, 0.8612)
##     No Information Rate : 0.3632
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.775
##
##  Mcnemar's Test P-Value : 0.0008716
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                    0.9465           0.7897          0.8024
## Specificity                    0.9491           0.9174          0.9109
## Pos Pred Value                 0.9138           0.8277          0.7961
## Neg Pred Value                 0.9688           0.8967          0.9140
## Prevalence                     0.3632           0.3344          0.3024
## Detection Rate                 0.3437           0.2640          0.2427
## Detection Prevalence           0.3761           0.3190          0.3048
## Balanced Accuracy              0.9478           0.8535          0.8566
```

Con los **datos imputados por KNN**:

```
modelo_lda_KNN <- lda(final.evaluation ~., data = datos_KNN[training_ids,])
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
pred_lda_KNN<- predict(modelo_lda_KNN, datos_KNN[-training_ids,])
tab_lda_KNN<-table(datos_KNN[-training_ids,]$final.evaluation, pred_lda_KNN$class,
                   dnn = c("Clase real", "Clase predicha"))
confusionMatrix(tab_lda_KNN)
```

```
## Confusion Matrix and Statistics
##
##            Clase predicha
## Clase real  not_recom recommend priority
##   not_recom       387       608      430
##   recommend       344      1031       61
##   priority        399        16      890
##
## Overall Statistics
##
##                Accuracy : 0.554
##                  95% CI : (0.5388, 0.5692)
##     No Information Rate : 0.3973
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3308
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                   0.34248           0.6230          0.6445
## Specificity                   0.65810           0.8387          0.8510
## Pos Pred Value                0.27158           0.7180          0.6820
## Neg Pred Value                0.72893           0.7714          0.8284
## Prevalence                    0.27124           0.3973          0.3315
## Detection Rate                0.09289           0.2475          0.2136
## Detection Prevalence          0.34205           0.3447          0.3133
## Balanced Accuracy             0.50029           0.7308          0.7477
```

Con los **datos imputados por KNN Weighted**:

```
modelo_lda_KNN_weighted <- lda(final.evaluation ~., data = datos_KNN_weighted[training_ids,])
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
pred_lda_KNN_weighted <- predict(modelo_lda_KNN_weighted, datos_KNN_weighted[-training_ids,])
tab_lda_KNN_weighted<-table(datos_KNN_weighted[-training_ids,]$final.evaluation, pred_lda_KNN_weight
confusionMatrix(tab_lda_KNN_weighted)
```

```
## Confusion Matrix and Statistics
##
##            Clase predicha
## Clase real  not_recom recommend priority
##   not_recom       396       599      432
##   recommend       344      1028       62
##   priority        397        18      890
##
## Overall Statistics
##
##                Accuracy : 0.5554
```

```
##                  95% CI : (0.5402, 0.5706)
##     No Information Rate : 0.3949
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.333
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                   0.34828           0.6249          0.6431
## Specificity                   0.65962           0.8390          0.8508
## Pos Pred Value                0.27751           0.7169          0.6820
## Neg Pred Value                0.72946           0.7742          0.8273
## Prevalence                    0.27292           0.3949          0.3322
## Detection Rate                0.09506           0.2468          0.2136
## Detection Prevalence          0.34253           0.3442          0.3133
## Balanced Accuracy             0.50395           0.7319          0.7469
```

Con los **datos imputados por PMM**:

```
modelo_lda_PMM <- lda(final.evaluation ~., data = datos_PMM[training_ids,])
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
pred_lda_PMM <- predict(modelo_lda_PMM, datos_PMM[-training_ids,])
tab_lda_PMM<-table(datos_PMM[-training_ids,]$final.evaluation, pred_lda_PMM$class,
                   dnn = c("Clase real", "Clase predicha"))
confusionMatrix(tab_lda_PMM)
```

```
## Confusion Matrix and Statistics
##
##            Clase predicha
## Clase real  not_recom recommend priority
##   not_recom       410       567      450
##   recommend       361       997       45
##   priority        412        14      910
##
## Overall Statistics
##
##                Accuracy : 0.5562
##                  95% CI : (0.5409, 0.5713)
##     No Information Rate : 0.3788
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3346
##
##  Mcnemar's Test P-Value : 9.553e-14
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                   0.34658           0.6318          0.6477
## Specificity                   0.65907           0.8431          0.8457
## Pos Pred Value                0.28732           0.7106          0.6811
## Neg Pred Value                0.71778           0.7897          0.8251
## Prevalence                    0.28397           0.3788          0.3373
## Detection Rate                0.09842           0.2393          0.2184
## Detection Prevalence          0.34253           0.3368          0.3207
```

```
## Balanced Accuracy             0.50282          0.7375          0.7467
```

Con los **datos originales**:

```
modelo_lda_orig <- lda(final.evaluation ~., data = datosorig[training_ids,])
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
pred_lda_orig <- predict(modelo_lda_MC, datosorig[-training_ids,])
tab_lda_orig<-table(datosorig[-training_ids,]$final.evaluation, pred_lda_orig$class,
                    dnn = c("Clase real", "Clase predicha"))
confusionMatrix(tab_lda_orig)
```

```
## Confusion Matrix and Statistics
##
##            Clase predicha
## Clase real  not_recom recommend priority
##   not_recom      1425         0        0
##   recommend         0      1305      102
##   priority          0       267     1067
##
## Overall Statistics
##
##                Accuracy : 0.9114
##                  95% CI : (0.9024, 0.9199)
##     No Information Rate : 0.3773
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8669
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                    1.0000           0.8302          0.9127
## Specificity                    1.0000           0.9607          0.9109
## Pos Pred Value                 1.0000           0.9275          0.7999
## Neg Pred Value                 1.0000           0.9032          0.9640
## Prevalence                     0.3421           0.3773          0.2806
## Detection Rate                 0.3421           0.3133          0.2561
## Detection Prevalence           0.3421           0.3377          0.3202
## Balanced Accuracy              1.0000           0.8954          0.9118
```

Con los **datos sin imputar**:

```
library(MASS)
modelo_lda_mis <- lda(final.evaluation ~., data = datos_mis[training_ids,])
pred_lda_mis <- predict(modelo_lda_mis, datos_mis[-training_ids,])
tab_lda_mis<-table(datos_mis[-training_ids,]$final.evaluation, pred_lda_mis$class,
                   dnn = c("Clase real", "Clase predicha"))
confusionMatrix(tab_lda_mis)
```

```
## Confusion Matrix and Statistics
##
##            Clase predicha
## Clase real  not_recom recommend priority
##   not_recom       247       338      268
##   recommend       213       616       26
##   priority        256         8      561
##
```

```
## Overall Statistics
##
##                Accuracy : 0.5622
##                  95% CI : (0.5426, 0.5816)
##     No Information Rate : 0.3798
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3433
##
##  Mcnemar's Test P-Value : 2.612e-08
##
## Statistics by Class:
##
##                      Class: not_recom Class: recommend Class: priority
## Sensitivity                   0.34497           0.6403          0.6561
## Specificity                   0.66648           0.8479          0.8427
## Pos Pred Value                0.28957           0.7205          0.6800
## Neg Pred Value                0.72083           0.7938          0.8279
## Prevalence                    0.28267           0.3798          0.3375
## Detection Rate                0.09751           0.2432          0.2215
## Detection Prevalence          0.33675           0.3375          0.3257
## Balanced Accuracy             0.50573           0.7441          0.7494
```

```
datoschamp<-read.csv("mushrooms.csv")
head(datoschamp)
```

```
##   class cap.shape cap.surface cap.color bruises odor gill.attachment
## 1     p         x           s         n       t    p               f
## 2     e         x           s         y       t    a               f
## 3     e         b           s         w       t    l               f
## 4     p         x           y         w       t    p               f
## 5     e         x           s         g       f    n               f
## 6     e         x           y         y       t    a               f
##   gill.spacing gill.size gill.color stalk.shape stalk.root
## 1            c         n          k           e          e
## 2            c         b          k           e          c
## 3            c         b          n           e          c
## 4            c         n          n           e          e
## 5            w         b          k           t          e
## 6            c         b          n           e          c
##   stalk.surface.above.ring stalk.surface.below.ring stalk.color.above.ring
## 1                        s                        s                       w
## 2                        s                        s                       w
## 3                        s                        s                       w
## 4                        s                        s                       w
## 5                        s                        s                       w
## 6                        s                        s                       w
##   stalk.color.below.ring veil.type veil.color ring.number ring.type
## 1                      w         p          w           o         p
## 2                      w         p          w           o         p
## 3                      w         p          w           o         p
## 4                      w         p          w           o         p
## 5                      w         p          w           o         e
## 6                      w         p          w           o         p
##   spore.print.color population habitat
## 1                 k          s       u
## 2                 n          n       g
## 3                 n          n       m
## 4                 k          s       u
## 5                 n          a       g
## 6                 k          n       g
```

```
dim(datoschamp)
```

```
## [1] 8124   23
```

```
datos<-datoschamp[,c(1,5,6,7,8,9,11,12,16,18,19,20,21,22)]
```

```
datos$class<-factor(datos$class)
datos$bruises<-factor(datos$bruises)
datos$odor<-factor(datos$odor)
datos$gill.attachment<-factor(datos$gill.attachment)
datos$gill.spacing<-factor(datos$gill.spacing)
datos$gill.size<-factor(datos$gill.size)
datos$stalk.shape<-factor(datos$stalk.shape)
datos$stalk.root<-factor(datos$stalk.root)
datos$stalk.color.below.ring<-factor(datos$stalk.color.below.ring)
datos$veil.color<-factor(datos$veil.color)
datos$ring.number<-factor(datos$ring.number)
datos$ring.type<-factor(datos$ring.type)
datos$spore.print.color<-factor(datos$spore.print.color)
datos$population<-factor(datos$population)
```

```
datosorig<-datos
```

```
table(datos$class)
```

```
##
##    e    p
## 4208 3916
```

```
table(datos$bruises)
```

```
##
##    f    t
## 4748 3376
```

```
table(datos$odor)
```

```
##
##    a    c    f    l    m    n    p    s    y
##  400  192 2160  400   36 3528  256  576  576
```

```
table(datos$gill.attachment)
```

```
##
##    a    f
##  210 7914
```

```
table(datos$gill.spacing)
```

```
##
##    c    w
## 6812 1312
```

```
table(datos$gill.size)
```

```
##
##    b    n
## 5612 2512
```

```
table(datos$stalk.shape)
```

```
##
##    e    t
## 3516 4608
```

```
table(datos$stalk.root)
```

```
##
##    ?    b    c    e    r
## 2480 3776  556 1120  192
```

```
table(datos$stalk.color.below.ring)
```

```
##
##    b    c    e    g    n    o    p    w    y
##  432   36   96  576  512  192 1872 4384   24
```

```
table(datos$veil.color)
```

```
##
##    n    o    w    y
##   96   96 7924    8
```

```
table(datos$ring.number)
```

```
##
##    n    o    t
```

```
##   36 7488  600
```

```
table(datos$ring.type)
```

```
##
##    e    f    l    n    p
## 2776   48 1296   36 3968
```

```
table(datos$spore.print.color)
```

```
##
##    b    h    k    n    o    r    u    w    y
##   48 1632 1872 1968   48   72   48 2388   48
```

```
table(datos$population)
```

```
##
##    a    c    n    s    v    y
##  384  340  400 1248 4040 1712
```

Se introducen datos faltantes:

```
set.seed(4)
indc<-sample(1:800,1)
indc
```

```
## [1] 504
```

```
ind<-sample(1:8124,indc)
indc2<-sample(1:800,1)
ind2<-sample(1:8124,indc2)
indc3<-sample(1:800,1)
ind3<-sample(1:8124,indc3)
indc4<-sample(1:800,1)
ind4<-sample(1:8124,indc4)
indc5<-sample(1:800,1)
ind5<-sample(1:8124,indc5)
indc6<-sample(1:800,1)
ind6<-sample(1:8124,indc6)
indc7<-sample(1:800,1)
ind7<-sample(1:8124,indc7)
indc8<-sample(1:800,1)
ind8<-sample(1:8124,indc8)
indc9<-sample(1:800,1)
ind9<-sample(1:8124,indc9)
indc10<-sample(1:800,1)
ind10<-sample(1:8124,indc10)
indc11<-sample(1:800,1)
ind11<-sample(1:8124,indc11)
indc12<-sample(1:800,1)
ind12<-sample(1:8124,indc12)
indc13<-sample(1:800,1)
ind13<-sample(1:8124,indc13)
indc14<-sample(1:800,1)
ind14<-sample(1:8124,indc14)


datos[ind,1]=NA
datos[ind2,2]=NA
datos[ind3,3]=NA
datos[ind4,4]=NA
datos[ind5,5]=NA
```

```
datos[ind6,6]=NA
datos[ind7,7]=NA
datos[ind8,8]=NA
datos[ind9,9]=NA
datos[ind10,10]=NA
datos[ind11,11]=NA
datos[ind12,12]=NA
datos[ind13,13]=NA
datos[ind14,14]=NA

head(datos)
```

```
##   class bruises odor gill.attachment gill.spacing gill.size stalk.shape
## 1     p       t    p               f            c         n           e
## 2  <NA>       t    a               f            c         b           e
## 3     e       t    l               f            c         b           e
## 4     p       t    p               f            c         n           e
## 5     e       f    n               f            w         b           t
## 6     e       t <NA>               f            c         b           e
##   stalk.root stalk.color.below.ring veil.color ring.number ring.type
## 1          e                      w          w           o      <NA>
## 2          c                   <NA>          w           o         p
## 3          c                      w          w           o         p
## 4          e                      w          w           o         p
## 5          e                      w          w           o      <NA>
## 6          c                      w          w           o         p
##   spore.print.color population
## 1                 k          s
## 2                 n          n
## 3                 n          n
## 4                 k          s
## 5                 n          a
## 6                 k          n
```

```
datos_mis<-datos
faltantes<-colSums(is.na(datos))
faltantes
```

```
##                  class                bruises                   odor
##                    504                     52                    502
##        gill.attachment          gill.spacing             gill.size
##                    266                    122                    690
##            stalk.shape           stalk.root stalk.color.below.ring
##                    386                    186                    655
##             veil.color           ring.number             ring.type
##                    334                    456                    756
##      spore.print.color             population
##                    104                    561
```

```
library(VIM)
```

```
## Warning: package 'VIM' was built under R version 4.1.2
```

```
## Loading required package: colorspace
```

```
## Loading required package: grid
```

```
## VIM is ready to use.
```

```
## Suggestions and bug-reports can be submitted at: https://github.com/statistikat/VIM/issues
```

```
##
```

```
## Attaching package: 'VIM'
```

```
## The following object is masked from 'package:datasets':
##
##     sleep
```

```
summary(datos)
```

```
##   class      bruises          odor      gill.attachment gill.spacing
##  e   :3929   f   :4722   n      :3294   a   : 200       c   :6709
##  p   :3691   t   :3350   f      :2043   f   :7658       w   :1293
##  NA's: 504   NA's:  52   s      : 546   NA's: 266       NA's: 122
##                          y      : 539
##                          a      : 374
##                          (Other): 826
##                          NA's   : 502
##  gill.size   stalk.shape stalk.root  stalk.color.below.ring veil.color
##  b   :5117   e   :3341   ?   :2429   w      :4041          n   :  90
##  n   :2317   t   :4397   b   :3685   p      :1733          o   :  91
##  NA's: 690   NA's: 386   c   : 543   g      : 522          w   :7601
##                          e   :1095   n      : 468          y   :   8
##                          r   : 186   b      : 382          NA's: 334
##                          NA's: 186   (Other): 323
##                                      NA's   : 655
##  ring.number ring.type   spore.print.color population
##  n   :  34   e   :2517   w      :2359      a   : 353
##  o   :7073   f   :  43   n      :1939      c   : 321
##  t   : 561   l   :1175   k      :1850      n   : 376
##  NA's: 456   n   :  33   h      :1613      s   :1168
##              p   :3600   r      :  71      v   :3758
##              NA's: 756   (Other): 188      y   :1587
##                          NA's   : 104      NA's: 561
```

Imputación

Imputación MC

```
datos_MC<-datos

getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}


imput<-function(df) {
  for (i in c(1:14)){
      index<-which(is.na(df[,i]))
      df[index,i]<-getmode(df[,i])
  }
  return(df)
}

datos_MC<-imput(datos_MC)
```

Se ve que efectivamente que se han imputado los datos y ya no hay ningún dato faltante:

```
colSums(is.na(datos_MC))
```

```
##                   class                 bruises                    odor
##                       0                       0                       0
```

```
##        gill.attachment           gill.spacing              gill.size
##                      0                      0                      0
##            stalk.shape             stalk.root stalk.color.below.ring
##                      0                      0                      0
##             veil.color            ring.number              ring.type
##                      0                      0                      0
##      spore.print.color             population
##                      0                      0
```

Porcentaje de error:

```
error_MC<-(8124-colSums(datos_MC[,1:14]==datosorig))/faltantes*100
error_MC
```

```
##                  class                bruises                   odor
##              44.642857              50.000000              53.386454
##        gill.attachment           gill.spacing              gill.size
##               3.759398              15.573770              28.260870
##            stalk.shape             stalk.root stalk.color.below.ring
##              45.336788              51.075269              47.633588
##             veil.color            ring.number              ring.type
##               3.293413               8.991228              51.322751
##      spore.print.color             population
##              72.115385              49.732620
```

```
table(datos_MC$class)
```

```
##
##    e    p
## 4433 3691
```

```
table(datos_MC$bruises)
```

```
##
##    f    t
## 4774 3350
```

```
table(datos_MC$odor)
```

```
##
##    a    c    f    l    m    n    p    s    y
##  374  181 2043  373   35 3796  237  546  539
```

```
table(datos_MC$gill.attachment)
```

```
##
##    a    f
##  200 7924
```

```
table(datos_MC$gill.spacing)
```

```
##
##    c    w
## 6831 1293
```

```
table(datos_MC$gill.size)
```

```
##
##    b    n
## 5807 2317
```

```
table(datos_MC$stalk.shape)
```

```
##
##    e    t
```

```
## 3341 4783
```

```
table(datos_MC$stalk.root)
```

```
##
##    ?    b    c    e    r
## 2429 3871  543 1095  186
```

```
table(datos_MC$stalk.color.below.ring)
```

```
##
##    b    c    e    g    n    o    p    w    y
##  382   35   91  522  468  178 1733 4696   19
```

```
table(datos_MC$veil.color)
```

```
##
##    n    o    w    y
##   90   91 7935    8
```

```
table(datos_MC$ring.number)
```

```
##
##    n    o    t
##   34 7529  561
```

```
table(datos_MC$ring.type)
```

```
##
##    e    f    l    n    p
## 2517   43 1175   33 4356
```

```
table(datos_MC$spore.print.color)
```

```
##
##    b    h    k    n    o    r    u    w    y
##   48 1613 1850 1939   46   71   46 2463   48
```

```
table(datos_MC$population)
```

```
##
##    a    c    n    s    v    y
##  353  321  376 1168 4319 1587
```

Imputación KNN:

```
library(VIM)
datos_KNN <-  kNN(datos, variable = c("class", "bruises", "odor", "gill.attachment","gill.spacing","
          "ring.type","spore.print.color","population" ), k = 5)
```

Se ve que efectivamente que se han imputado los datos y ya no hay ningún dato faltante:

```
colSums(is.na(datos_KNN))
```

```
##                      class                    bruises
##                          0                          0
##                       odor            gill.attachment
##                          0                          0
##              gill.spacing                  gill.size
##                          0                          0
##               stalk.shape                 stalk.root
##                          0                          0
##    stalk.color.below.ring                 veil.color
##                          0                          0
```

```
##                ring.number                   ring.type
##                          0                           0
##          spore.print.color                  population
##                          0                           0
##                  class_imp                bruises_imp
##                          0                           0
##                   odor_imp         gill.attachment_imp
##                          0                           0
##           gill.spacing_imp               gill.size_imp
##                          0                           0
##            stalk.shape_imp              stalk.root_imp
##                          0                           0
## stalk.color.below.ring_imp              veil.color_imp
##                          0                           0
##            ring.number_imp               ring.type_imp
##                          0                           0
##      spore.print.color_imp              population_imp
##                          0                           0
```

Porcentaje de error:

```
error_KNN<-(8124-colSums(datos_KNN[,1:14]==datosorig))/faltantes*100
error_KNN
```

```
##                  class                bruises                   odor
##              0.0000000              0.0000000             19.5219124
##        gill.attachment           gill.spacing              gill.size
##              0.0000000              1.6393443              0.0000000
##             stalk.shape             stalk.root stalk.color.below.ring
##              0.2590674              1.0752688             37.5572519
##              veil.color            ring.number              ring.type
##              2.0958084              0.0000000              0.0000000
##      spore.print.color             population
##             27.8846154             33.5115865
```

Imputacion KNN Weighted:

```
datos_KNN_weighted <-  kNN(datos, variable = c("class", "bruises", "odor", "gill.attachment","gill.s
"ring.type","spore.print.color","population" ), k = 5, weightDist=TRUE)
```

Se ve que efectivamente que se han imputado los datos y ya no hay ningún dato faltante:

```
colSums(is.na(datos_KNN_weighted))
```

```
##                      class                    bruises
##                          0                          0
##                       odor             gill.attachment
##                          0                          0
##               gill.spacing                  gill.size
##                          0                          0
##                stalk.shape                 stalk.root
##                          0                          0
##     stalk.color.below.ring                 veil.color
##                          0                          0
##                ring.number                  ring.type
##                          0                          0
##          spore.print.color                 population
##                          0                          0
##                  class_imp                bruises_imp
##                          0                          0
```

```
##                   odor_imp        gill.attachment_imp
##                          0                          0
##           gill.spacing_imp              gill.size_imp
##                          0                          0
##            stalk.shape_imp             stalk.root_imp
##                          0                          0
## stalk.color.below.ring_imp             veil.color_imp
##                          0                          0
##            ring.number_imp              ring.type_imp
##                          0                          0
##      spore.print.color_imp             population_imp
##                          0                          0
```

Porcentaje de error:

```
error_KNN_weighted<-(8124-colSums(datos_KNN_weighted[,1:14]==datosorig))/faltantes*100
error_KNN_weighted
```

```
##                  class                bruises                   odor
##              0.0000000              0.0000000             18.5258964
##        gill.attachment           gill.spacing              gill.size
##              0.0000000              1.6393443              0.0000000
##             stalk.shape             stalk.root stalk.color.below.ring
##              0.2590674              1.0752688             38.0152672
##              veil.color            ring.number              ring.type
##              2.0958084              0.0000000              0.0000000
##      spore.print.color             population
##             27.8846154             33.3333333
```

Imputación PMM

```
library(mice)
```

```
## Warning: package 'mice' was built under R version 4.1.2
```

```
##
## Attaching package: 'mice'
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```
imp_multi <- mice(datos, m = 5, method = "pmm")  # Impute missing values multiple times
```

```
##
##  iter imp variable
##   1   1  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   1   2  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   1   3  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   1   4  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   1   5  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   2   1  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   2   2  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   2   3  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   2   4  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   2   5  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   3   1  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   3   2  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
```

```
##   3   3  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   3   4  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   3   5  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   4   1  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   4   2  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   4   3  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   4   4  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   4   5  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   5   1  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   5   2  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   5   3  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   5   4  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
##   5   5  class  bruises  odor  gill.attachment  gill.spacing  gill.size  stalk.shape  stalk.root
```

```
## Warning: Number of logged events: 241
```

```
datos_PMM<-mice::complete(imp_multi)
```

Se ve que efectivamente los datos se han imputado y no hay ningún dato faltante:

```
colSums(is.na(datos_PMM))
```

```
##                   class                 bruises                    odor
##                       0                       0                       0
##         gill.attachment           gill.spacing               gill.size
##                       0                       0                       0
##             stalk.shape             stalk.root stalk.color.below.ring
##                       0                       0                       0
##              veil.color            ring.number               ring.type
##                       0                       0                       0
##       spore.print.color              population
##                       0                       0
```

Porcentaje de error:

```
error_PMM<-(8124-colSums(datos_PMM[,1:14]==datosorig))/faltantes*100
error_PMM
```

```
##                   class                 bruises                    odor
##               0.3968254               1.9230769              27.4900398
##         gill.attachment           gill.spacing               gill.size
##               0.0000000               1.6393443               0.2898551
##             stalk.shape             stalk.root stalk.color.below.ring
##               1.2953368              10.2150538              45.0381679
##              veil.color            ring.number               ring.type
##               1.7964072               0.0000000               3.9682540
##       spore.print.color              population
##              25.9615385              43.3155080
```

Clasificación

Naive Bayes

Con los **datos imputados por MC**:

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.1.2
```

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.1.2
```

```
## Loading required package: lattice
```

```
set.seed(4)
training_ids<-createDataPartition(datoschamp$class, p=0.67, list=F)
```

```
model_MC<-naiveBayes(class~.,datos_MC[training_ids,])
pred_MC <- predict(model_MC, datos_MC[-training_ids,])
tab_MC<-table(datos_MC[-training_ids,]$class, pred_MC,
              dnn=c("Actual","Predicha"))
confusionMatrix(tab_MC)
```

```
## Confusion Matrix and Statistics
##
##       Predicha
## Actual    e    p
##      e 1371   76
##      p   84 1149
##
##                Accuracy : 0.9403
##                  95% CI : (0.9307, 0.949)
##     No Information Rate : 0.5429
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8798
##
##  Mcnemar's Test P-Value : 0.58
##
##             Sensitivity : 0.9423
##             Specificity : 0.9380
##          Pos Pred Value : 0.9475
##          Neg Pred Value : 0.9319
##              Prevalence : 0.5429
##          Detection Rate : 0.5116
##    Detection Prevalence : 0.5399
##       Balanced Accuracy : 0.9401
##
##        'Positive' Class : e
##
```

Con los **datos imputados por KNN**:

```
model_KNN<-naiveBayes(class~.,datos_KNN[training_ids,])
pred_KNN<-predict(model_KNN, datos_KNN[-training_ids,])
tab_KNN<-table(datos_KNN[-training_ids,]$class, pred_KNN,
               dnn=c("Actual","Predicha"))
confusionMatrix(tab_KNN)
```

```
## Confusion Matrix and Statistics
##
##       Predicha
## Actual    e    p
##      e 1367   21
##      p   39 1253
##
##                Accuracy : 0.9776
##                  95% CI : (0.9713, 0.9829)
##     No Information Rate : 0.5246
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9551
```

```
##
##  Mcnemar's Test P-Value : 0.02819
##
##             Sensitivity : 0.9723
##             Specificity : 0.9835
##          Pos Pred Value : 0.9849
##          Neg Pred Value : 0.9698
##              Prevalence : 0.5246
##          Detection Rate : 0.5101
##    Detection Prevalence : 0.5179
##       Balanced Accuracy : 0.9779
##
##        'Positive' Class : e
##
```

Con los **datos imputados por KNN Weighted**:

```
model_KNN_weighted<-naiveBayes(class~.,
                datos_KNN_weighted[training_ids,])
pred_KNN_weighted <- predict(model_KNN_weighted,
                datos_KNN_weighted[-training_ids,])
tab_KNN_weighted<-table(datos_KNN_weighted[-training_ids,]$class,
                pred_KNN_weighted, dnn=c("Actual","Predicha"))
confusionMatrix(tab_KNN_weighted)
```

```
## Confusion Matrix and Statistics
##
##       Predicha
## Actual    e    p
##      e 1367   21
##      p   39 1253
##
##                Accuracy : 0.9776
##                  95% CI : (0.9713, 0.9829)
##     No Information Rate : 0.5246
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9551
##
##  Mcnemar's Test P-Value : 0.02819
##
##             Sensitivity : 0.9723
##             Specificity : 0.9835
##          Pos Pred Value : 0.9849
##          Neg Pred Value : 0.9698
##              Prevalence : 0.5246
##          Detection Rate : 0.5101
##    Detection Prevalence : 0.5179
##       Balanced Accuracy : 0.9779
##
##        'Positive' Class : e
##
```

Con los **datos imputados por PMM**:

```
model_PMM<-naiveBayes(class~.,datos_PMM[training_ids,])
pred_PMM <- predict(model_PMM, datos_PMM[-training_ids,])
tab_PMM<-table(datos_PMM[-training_ids,]$class, pred_PMM,
               dnn=c("Actual","Predicha"))
confusionMatrix(tab_PMM)
```

```
## Confusion Matrix and Statistics
##
##       Predicha
## Actual    e    p
##      e 1365   21
##      p   46 1248
##
##                Accuracy : 0.975
##                  95% CI : (0.9684, 0.9806)
##     No Information Rate : 0.5265
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9499
##
##  Mcnemar's Test P-Value : 0.003367
##
##             Sensitivity : 0.9674
##             Specificity : 0.9835
##          Pos Pred Value : 0.9848
##          Neg Pred Value : 0.9645
##              Prevalence : 0.5265
##          Detection Rate : 0.5093
##    Detection Prevalence : 0.5172
##       Balanced Accuracy : 0.9754
##
##        'Positive' Class : e
##
```

Con los **datos originales**:

```
model_orig<-naiveBayes(class~.,datosorig[training_ids,])
pred_orig<- predict(model_orig, datosorig[-training_ids,])
tab_orig<-table(datosorig[-training_ids,]$class, pred_orig,
                dnn=c("Actual","Predicha"))
confusionMatrix(tab_orig)
```

```
## Confusion Matrix and Statistics
##
##       Predicha
## Actual    e    p
##      e 1367   21
##      p   49 1243
##
##                Accuracy : 0.9739
##                  95% CI : (0.9671, 0.9796)
##     No Information Rate : 0.5284
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9477
##
##  Mcnemar's Test P-Value : 0.00125
##
##             Sensitivity : 0.9654
##             Specificity : 0.9834
##          Pos Pred Value : 0.9849
##          Neg Pred Value : 0.9621
##              Prevalence : 0.5284
##          Detection Rate : 0.5101
##    Detection Prevalence : 0.5179
```

```
##       Balanced Accuracy : 0.9744
##
##        'Positive' Class : e
##
```

Con los **datos sin imputar**:

```
model_mis<-naiveBayes(class~.,datos_mis[training_ids,])
pred_mis <- predict(model_mis, datos_mis[-training_ids,])
tab_mis<-table(datos_mis[-training_ids,]$class, pred_mis,
               dnn=c("Actual","Predicha"))
confusionMatrix(tab_mis)
```

```
## Confusion Matrix and Statistics
##
##       Predicha
## Actual    e    p
##      e 1274   18
##      p   52 1181
##
##                Accuracy : 0.9723
##                  95% CI : (0.9651, 0.9783)
##     No Information Rate : 0.5251
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9445
##
##  Mcnemar's Test P-Value : 8.005e-05
##
##             Sensitivity : 0.9608
##             Specificity : 0.9850
##          Pos Pred Value : 0.9861
##          Neg Pred Value : 0.9578
##              Prevalence : 0.5251
##          Detection Rate : 0.5046
##    Detection Prevalence : 0.5117
##       Balanced Accuracy : 0.9729
##
##        'Positive' Class : e
##
```

SVM

Con los **datos imputados por MC**:

```
library(e1071)
set.seed(4)
modelo_svm_MC <- svm(class ~., datos_MC[training_ids,],
          kernel = "linear", cost = 10, scale = FALSE)
pred_svm_MC<-predict(modelo_svm_MC,datos_MC[-training_ids,])
tab_svm_MC<-table(datos_MC[-training_ids,]$class,
                pred_svm_MC, dnn=c("Actual","Predicha"))
confusionMatrix(tab_svm_MC)
```

```
## Confusion Matrix and Statistics
##
##       Predicha
## Actual    e    p
##      e 1380   67
##      p   15 1218
```

```
##
##                Accuracy : 0.9694
##                  95% CI : (0.9622, 0.9756)
##     No Information Rate : 0.5205
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9386
##
##  Mcnemar's Test P-Value : 1.781e-08
##
##             Sensitivity : 0.9892
##             Specificity : 0.9479
##          Pos Pred Value : 0.9537
##          Neg Pred Value : 0.9878
##              Prevalence : 0.5205
##          Detection Rate : 0.5149
##    Detection Prevalence : 0.5399
##       Balanced Accuracy : 0.9686
##
##        'Positive' Class : e
##
```

Con los **datos imputados por KNN**:

```
modelo_svm_KNN<- svm(class ~., datos_KNN[training_ids,],
            kernel = "linear", cost = 10, scale = FALSE)
pred_svm_KNN<-predict(modelo_svm_KNN,datos_KNN[-training_ids,])
tab_svm_KNN<-table(datos_KNN[-training_ids,]$class,
          pred_svm_KNN, dnn=c("Actual","Predicha"))
confusionMatrix(tab_svm_KNN)
```

```
## Confusion Matrix and Statistics
##
##       Predicha
## Actual    e    p
##      e 1388    0
##      p    0 1292
##
##                Accuracy : 1
##                  95% CI : (0.9986, 1)
##     No Information Rate : 0.5179
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5179
##          Detection Rate : 0.5179
##    Detection Prevalence : 0.5179
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : e
##
```

Con los **datos imputados por WKNN**:

```
modelo_svm_KNN_weighted <- svm(class ~.,
            datos_KNN_weighted[training_ids,],
            kernel = "linear", cost = 10, scale = FALSE)
pred_svm_KNN_weighted<-predict(modelo_svm_KNN_weighted,
                  datos_KNN_weighted[-training_ids,])
tab_svm_KNN_weighted<-table(datos_KNN_weighted[-training_ids,]$class,
            pred_svm_KNN_weighted, dnn=c("Actual","Predicha"))
confusionMatrix(tab_svm_KNN_weighted)
```

```
## Confusion Matrix and Statistics
##
##       Predicha
## Actual    e    p
##      e 1388    0
##      p    0 1292
##
##                Accuracy : 1
##                  95% CI : (0.9986, 1)
##     No Information Rate : 0.5179
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5179
##          Detection Rate : 0.5179
##    Detection Prevalence : 0.5179
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : e
##
```

Con los **datos imputados por PMM**:

```
modelo_svm_PMM <- svm(class ~., datos_PMM[training_ids,],
              kernel = "linear", cost = 10, scale = FALSE)
pred_svm_PMM<-predict(modelo_svm_PMM,datos_PMM[-training_ids,])
tab_svm_PMM<-table(datos_PMM[-training_ids,]$class, pred_svm_PMM,
          dnn=c("Actual","Predicha"))
confusionMatrix(tab_svm_PMM)
```

```
## Confusion Matrix and Statistics
##
##       Predicha
## Actual    e    p
##      e 1386    0
##      p    6 1288
##
##                Accuracy : 0.9978
##                  95% CI : (0.9951, 0.9992)
##     No Information Rate : 0.5194
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9955
```

```
##
##  Mcnemar's Test P-Value : 0.04123
##
##             Sensitivity : 0.9957
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9954
##              Prevalence : 0.5194
##          Detection Rate : 0.5172
##    Detection Prevalence : 0.5172
##       Balanced Accuracy : 0.9978
##
##        'Positive' Class : e
##
```

Con los **datos originales**:

```
modelo_svm_orig <- svm(class~., datosorig[training_ids,],
            kernel = "linear", cost = 10, scale = FALSE)
pred_svm_orig<-predict(modelo_svm_orig,datosorig[-training_ids,])
tab_svm_orig<-table(datosorig[-training_ids,]$class,
            pred_svm_orig, dnn=c("Actual","Predicha"))
confusionMatrix(tab_svm_orig)
```

```
## Confusion Matrix and Statistics
##
##       Predicha
## Actual    e    p
##      e 1388    0
##      p    0 1292
##
##                Accuracy : 1
##                  95% CI : (0.9986, 1)
##     No Information Rate : 0.5179
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5179
##          Detection Rate : 0.5179
##    Detection Prevalence : 0.5179
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : e
##
```

Con los **datos sin imputar**:

```
modelo_svm_mis<- svm(class ~., datos_mis[training_ids,],
            kernel = "linear", cost = 10, scale = FALSE)
pred_svm_mis<-predict(modelo_svm_mis,datos_mis[-training_ids,])
indicese<-which(pred_svm_mis=="e")
indicesp<-which(pred_svm_mis=="p")
indicespred<-c(indicese,indicesp)
```

```
tab_svm_mis<-table(datos_mis[-training_ids,]$class[indicespred]==pred_svm_mis)
tab_svm_mis
```

```
##
## FALSE  TRUE
##   566   684
```

```
#tab_svm_mis<-table(datos_mis[indicespred,]$class pred_svm_mis, dnn=c("Actual","Predicha"))
#confusionMatrix(tab_svm_mis)
```

Análisis Discriminante

Con los **datos imputados por MC**:

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 4.1.2
```

```
modelo_lda_MC <- lda(class ~., data = datos_MC[training_ids,])
pred_lda_MC <- predict(modelo_lda_MC, datos_MC[-training_ids,])
tab_lda_MC<-table(datos_MC[-training_ids,]$class, pred_lda_MC$class,
             dnn = c("Clase real", "Clase predicha"))
confusionMatrix(tab_lda_MC)
```

```
## Confusion Matrix and Statistics
##
##           Clase predicha
## Clase real    e    p
##          e 1387   60
##          p   15 1218
##
##                Accuracy : 0.972
##                  95% CI : (0.965, 0.9779)
##     No Information Rate : 0.5231
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9438
##
##  Mcnemar's Test P-Value : 3.761e-07
##
##             Sensitivity : 0.9893
##             Specificity : 0.9531
##          Pos Pred Value : 0.9585
##          Neg Pred Value : 0.9878
##              Prevalence : 0.5231
##          Detection Rate : 0.5175
##    Detection Prevalence : 0.5399
##       Balanced Accuracy : 0.9712
##
##        'Positive' Class : e
##
```

Con los **datos imputados por KNN**:

```
modelo_lda_KNN<- lda(class ~., data = datos_KNN[training_ids,])
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
pred_lda_KNN<- predict(modelo_lda_KNN, datos_KNN[-training_ids,])
tab_lda_KNN<-table(datos_KNN[-training_ids,]$class, pred_lda_KNN$class,
             dnn = c("Clase real", "Clase predicha"))
confusionMatrix(tab_lda_KNN)
```

```
## Confusion Matrix and Statistics
##
##           Clase predicha
## Clase real    e    p
##          e 1388    0
##          p    0 1292
##
##                Accuracy : 1
##                  95% CI : (0.9986, 1)
##     No Information Rate : 0.5179
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5179
##          Detection Rate : 0.5179
##    Detection Prevalence : 0.5179
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : e
##
```

Con los **datos imputados por WKNN**:

```
modelo_lda_KNN_weighted <- lda(class ~., data = datos_KNN_weighted[training_ids,])
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
pred_lda_KNN_weighted <- predict(modelo_lda_KNN_weighted,
                      datos_KNN_weighted[-training_ids,])
tab_lda_KNN_weighted<-table(datos_KNN_weighted[-training_ids,]$class,
  pred_lda_KNN_weighted$class, dnn = c("Clase real", "Clase predicha"))
confusionMatrix(tab_lda_KNN_weighted)
```

```
## Confusion Matrix and Statistics
##
##           Clase predicha
## Clase real    e    p
##          e 1388    0
##          p    0 1292
##
##                Accuracy : 1
##                  95% CI : (0.9986, 1)
##     No Information Rate : 0.5179
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
```

```
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5179
##          Detection Rate : 0.5179
##    Detection Prevalence : 0.5179
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : e
##
```

Con los **datos imputados por PMM**:

```
modelo_lda_PMM <- lda(class ~., data = datos_PMM[training_ids,])
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
pred_lda_PMM <- predict(modelo_lda_PMM, datos_PMM[-training_ids,])
tab_lda_PMM<-table(datos_PMM[-training_ids,]$class, pred_lda_PMM$class,
            dnn = c("Clase real", "Clase predicha"))
confusionMatrix(tab_lda_PMM)
```

```
## Confusion Matrix and Statistics
##
##           Clase predicha
## Clase real    e    p
##          e 1378    8
##          p    9 1285
##
##                Accuracy : 0.9937
##                  95% CI : (0.9899, 0.9963)
##     No Information Rate : 0.5175
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9873
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9935
##             Specificity : 0.9938
##          Pos Pred Value : 0.9942
##          Neg Pred Value : 0.9930
##              Prevalence : 0.5175
##          Detection Rate : 0.5142
##    Detection Prevalence : 0.5172
##       Balanced Accuracy : 0.9937
##
##        'Positive' Class : e
##
```

Con los **datos originales**:

```
modelo_lda_orig <- lda(class ~., data = datosorig[training_ids,])
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
pred_lda_orig <- predict(modelo_lda_MC, datosorig[-training_ids,])
tab_lda_orig<-table(datosorig[-training_ids,]$class, pred_lda_orig$class,
            dnn = c("Clase real", "Clase predicha"))
confusionMatrix(tab_lda_orig)
```

```
## Confusion Matrix and Statistics
##
##           Clase predicha
```

```
## Clase real    e    p
##          e 1388    0
##          p    2 1290
##
##                Accuracy : 0.9993
##                  95% CI : (0.9973, 0.9999)
##     No Information Rate : 0.5187
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9985
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 0.9986
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9985
##              Prevalence : 0.5187
##          Detection Rate : 0.5179
##    Detection Prevalence : 0.5179
##       Balanced Accuracy : 0.9993
##
##        'Positive' Class : e
##
```

Con los **datos sin imputar**:

```
modelo_lda_mis <- lda(class ~., data = datos_mis[training_ids,])
pred_lda_mis <- predict(modelo_lda_mis, datos_mis[-training_ids,])
tab_lda_mis<-table(datos_mis[-training_ids,]$class, pred_lda_mis$class,
                dnn = c("Clase real", "Clase predicha"))
confusionMatrix(tab_lda_mis)
```

```
## Confusion Matrix and Statistics
##
##           Clase predicha
## Clase real   e   p
##          e 718   0
##          p   0 618
##
##                Accuracy : 1
##                  95% CI : (0.9972, 1)
##     No Information Rate : 0.5374
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5374
##          Detection Rate : 0.5374
##    Detection Prevalence : 0.5374
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : e
```

##