FACULTAD DE MATEMÁTICAS

DOBLE GRADO EN MATEMÁTICAS Y ESTADÍSTICA

Análisis del mercado de los juegos de mesa en perspectiva matemático-computacional

Trabajo Fin de Grado presentado por José Antonio Rodríguez Gallego, siendo el tutor del mismo Luis Valencia Cabrera.

Sevilla, Julio de 2021

FACULTAD DE MATEMÁTICAS

DOBLE GRADO EN MATEMÁTICAS Y ESTADÍSTICA

TRABAJO FIN DE GRADO

CURSO ACADÉMICO [2020-2021]

TÍTULO: ANÁLISIS DEL MERCADO DE LOS JUEGOS DE MESA EN PERSPECTIVA MATEMÁTICO-COMPUTACIONAL

AUTOR: JOSÉ ANTONIO RODRÍGUEZ GALLEGO

TUTOR: DR. D. LUIS VALENCIA CABRERA

DEPARTAMENTO: Ciencias de la Computación e Inteligencia Artificial

ÁREA DE CONOCIMIENTO: Ciencias de la Computación e Inteligencia Artificial

RESUMEN:

El mercado de los juegos de mesa ha presentado, a lo largo de los últimos años, un desarrollo sin precedentes, convirtiéndose en un engranaje de los mercados internacionales. Teniendo esto en cuenta, resulta informativo a la par que interesante realizar un análisis en profundidad del mismo, buscando patrones y basándonos en ellos para construir modelos de predicción y obtener información inferencial, como saber cuáles son las tendencias del mercado en lo referido a temáticas, público objetivo o valoraciones de los usuarios. Con esas ideas en mente, el siguiente proyecto intenta dar respuestas a algunas de dichas preguntas, partiendo de una potente base teórica estadística y computacional, que más adelante se aplicará en diferentes conjuntos de datos, proporcionándonos información de variada índole relativa al comportamiento de este sector que hemos decidido estudiar.

Aplicaremos multitud de herramientas estudiadas a lo largo del currículo académico para poder construir modelos de predicción, y comprobaremos recurriendo a multitud de métricas la bondad y validez de los resultados obtenidos. Además, desarrollaremos en detalle un ejemplo de acceso a API, comentando sus fortalezas y debilidades, seguido de la extracción de los datos, acompañada del tratamiento y preparación asociados para poder trabajar con ellos. Cada paso que demos estos ámbitos se verá acompañado de tablas, gráficas y fragmentos de código que facilitarán la comprensión del trabajo aquí expuesto.

Adicionalmente, aprovecharemos el gran número de comentarios obtenidos en el punto previo para aplicar diversas técnicas de Procesamiento del Lenguaje Natural, incluyendo entre otros, el modelo de espacio vectorial o diferentes formas de abordar el análisis del sentimiento

Por último, en el capítulo 6 expondremos las principales conclusiones extraídas sobre el trabajo, agrupando los resultados más remarcables del proyecto en un único lugar, sirviendo de esta forma como referencia para aquella persona que esté interesada principalmente en los resultados.

ABSTRACT:

The market of board games has experienced, throughout the last year, a groundbreaking growth, becoming one of the gears of the international markets. Taking this into account, it is both enlightening and interesting to make a deep analysis in this area, looking for patterns and using them in order to develop prediction models, obtaining inferential information, such as knowing which are the market trends in what refers to the topics, target audience or the users' ratings. Keeping these ideas in mind, the present project intends to answer some of those questions, starting from a powerful theoretical basis in both statistics and computation, which later will be applied to different datasets, providing us with varied information related to the behavior of the sector which we have decided to analyze.

We will apply some of the different tools studied along the degree towards the development of prediction models, and then we will test their soundness and validity using several metrics. Moreover, we will deeply develop an example of access to an API, discussing its strengths and weaknesses, followed by the extraction of the data, which will be done with the proper treatment and preparation to make them ready for further steps. Each of those steps will be done with the aid of graphs, tables and code snippets that will make it easier to understand the project.

Furthermore, we will take advantage of the great number of comments obtained in the previous step to apply different technics of the field of Natural Language Processing, being included, among others, the vectorial space model or different approaches to Sentiment Analysis.

Finally, we will end the project showing the main results obtained and lessons learned, grouping the most remarkable results in one place, serving as a reference to those who are mostly interested in the answers obtained.

TÉRMINOS CLAVE: ANÁLISIS DEL MERCADO DE LOS JUEGOS DE MESA; ANÁLISIS DE DATOS DE BOARDGAMEGEEK; SENTIMENT ANALYSIS; PROCESAMIENTO DEL LENGUAJE NATURAL (NLP); APRENDIZAJE AUTOMÁTICO (MACHINE LEARNING)

ÍNDICE

Relación de Figuras

1 INTRODUCCIÓN

La memoria aquí presente buscar describir los aspectos fundamentales del Trabajo Fin de Grado realizado a lo largo de este curso 2020-2021, centrado en el mercado de los juegos de mesa, abordado desde una perspectiva que combine la visión matemático-estadística adquirida a lo largo del currículo académico con herramientas y técnicas de las ciencias de la computación y la Inteligencia Artificial.

En este primer capítulo explicaremos el contexto en el que se ubicará el proyecto, se clarificarán los objetivos principales que se buscan y se indicará la estructura general de la memoria, indicando el orden en el que se presentarán los diferentes contenidos que desglosaremos en las siguientes páginas.

Primeramente expondremos una breve motivación del trabajo, aportando algunos datos para explicar la importancia del sector en el que hemos centrado nuestra atención. Siguiendo esta idea, presentaremos cifras extraídas de multitud de fuentes para justificar no solo la relevancia del trabajo, sino también su viabilidad al existir multitud de fuentes que se pueden consultar para llevar a cabo el análisis estadístico que se planea hacer.

1.1 Motivación

Desde pequeños, ya sea con nuestros familiares o con nuestros amigos, hemos jugado a juegos de mesa de todo tipo: desde el parchís a la oca, pasando por todos los juegos de cartas que se nos pueden pasar por la cabeza. Sin embargo, si sentimos curiosidad por el tema y realizamos una representación del número de juegos publicados por año, como muestra la Figura 1.1, nos damos cuenta rápidamente de que los hábitos relacionados con los juegos de mesa han cambiado drásticamente, incrementándose sustancialmente la oferta de productos posibles en este sector.

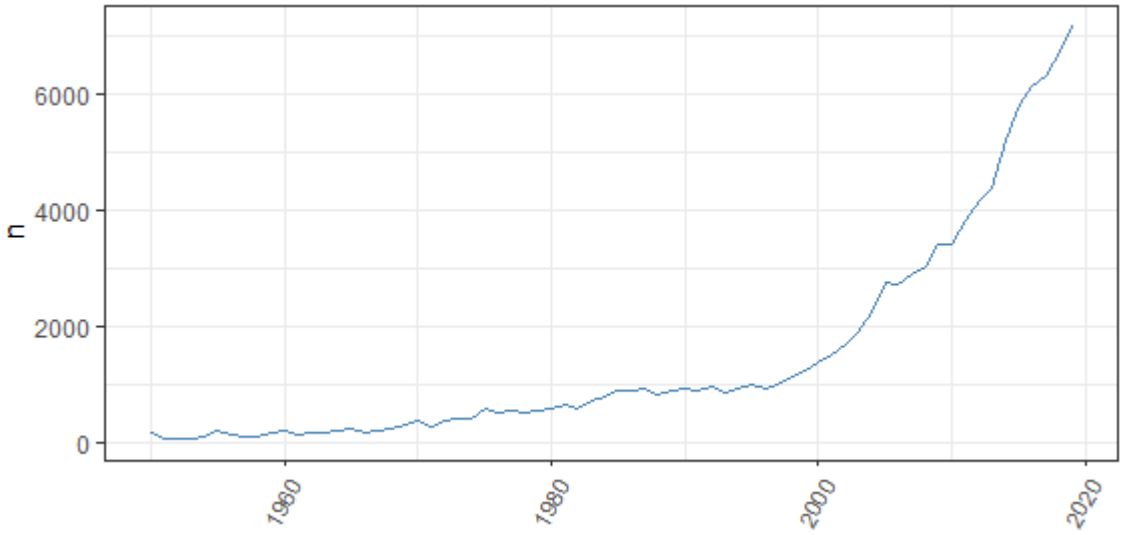

Figura 1.1. Publicaciones de juegos de mesa entre 1950 y 2019

Fuente: Elaboración propia a partir de datos de boardgamegeek.com

Es cierto que se sigue jugando a los clásicos ya mencionados, pero hoy en día existe un extenso abanico de posibilidades que se abre ante nosotros en lo que a juegos posibles se refiere, pudiendo uno encontrar juegos de prácticamente cualquier temática, con mecánicas adaptadas para todos los públicos.

Cabe destacar, sin embargo, que al tiempo que nuevos juegos de mesa se lanzan diariamente, los tradicionales cada vez ganan más y más popularidad en plataformas digitales, llegando a disfrutar de la misma fama que otros conocidos *e-sports* en plataformas de *streaming* como *Twitch.com*. A modo de ejemplo, podemos encontrar el ajedrez entre las 35 categorías más populares de la plataforma mencionada [27, 36]. Siguiendo con esta idea y extendiéndola, en [31] podemos comprobar el número de visitas a páginas reconocidas como *chess.com* o *cardgames.io*, y ver que, efectivamente, tienen influencia en la red a día de hoy. Cabe destacar también la aparición dentro del listado de *boardgamearena* [25], plataforma que permite jugar a juegos de mesa modernos, dividiéndolos entre juegos gratuitos y juegos que necesitan del pago de una suscripción, siendo otro este otro ejemplo de la expansión que han sufrido los juegos de mesa a través de multitud de plataformas web.

Es en este contexto de expansión del sector donde se ubica nuestro interés, donde parece interesante realizar un estudio del mercado, encontrándonos de esta forma con un punto en el que aplicar de forma clara muchas de las herramientas estadísticas y computacionales vistas a lo largo del currículo académico, así como otras nuevas en las

que hemos podido profundizar ampliamente y que desglosaremos a lo largo de los siguientes capítulos.

1.2 El mercado de los juegos de mesa

Ver el número de juegos publicados a lo largo de los años podría no ser suficiente para certificar la importancia del mercado de los juegos de mesa pero, otros hechos y cifras, como las que vamos a exponer ahora, acreditarán con seguridad la relevancia del sector hoy en día.

En palabras del análisis realizado por *Grand View Search* [28], en el que detallan la importancia económica de juegos más clásicos como el ajedrez, el *Monopoly* o el *Scrabble*, solo con un pequeño conjunto de títulos de esta índole nos encontramos con que en 2018 el tamaño estimado del mercado era de 11.95 mil millones de dólares americanos, y que se estimaba un crecimiento de dicho mercado durante los próximos años. Es llamativo comprobar cómo, en dicho informe, la mayor parte de la importancia del sector es debida a estos juegos tradicionales ya mencionados, también porque, como ya comentamos, tienen una parte importante de su peso en formato digital.

Por acudir a otro ejemplo de mayor interés para nosotros, veamos por ejemplo el juego *Colonos de Catán*, lanzado en 1995 y que, como indican en la página de venta oficial del producto [26], ha llegado a vender más de 2 millones de copias en Europa y América, convirtiéndose en un juego obligatorio para todas las personas y asociaciones que disfrutan de este pasatiempo. Siguiendo el ejemplo de este juego nos podemos encontrar con muchos otros juegos: *Carcassonne*, *Aventureros al Tren*, *Mysterium*…

Cabe destacar, por otro lado, que el mercado de los juegos de mesa repercute en muchos otros, pues desde los artistas que decoran las cajas e ilustraciones de los juegos hasta las empresas de fabricación que han de producir los diversos componentes de los juegos o aquellas encargadas de distribuirlos, traducirlos, etc., el sector proporciona fluidez a todas ellas, teniendo un impacto mucho mayor del que uno podría esperar en un primer momento. Además, resulta llamativo, y un hecho a tener en cuenta también, el número de cafeterías y bares que centran su temática en los juegos de mesa hoy en día. Este número se va incrementando poco a poco, habiendo varios ejemplos de este suceso en la ciudad de Sevilla (*Dragón Verde, Kame House, Arcade Bar, Star Bar*…) y otros de mayores dimensiones como, por ejemplo, en Londres, donde el *Draughts Café* dispone ya de aproximadamente 850 juegos de mesa en su ludoteca [28].

En definitiva, queda claro que el sector de los juegos de mesa es uno que hoy en día se encuentra activo y espera prosperar, que tiene valor real dentro del mercado y que puede estar presente en el día a día de muchos de nosotros. Ahora, con el objetivo de obtener más información sobre el mismo, abordamos este proyecto, tratando de acudir a fuentes específicas del sector para analizar aspectos relacionados con el éxito de los juegos y su percepción en la comunidad.

1.3 Objetivos del proyecto

Explicamos en esta sección con un mayor detalle los objetivos que buscamos lograr a lo largo del trabajo, intentando agruparlos de forma semejante a como se expondrán.

Primeramente, buscamos obtener un conjunto de datos amplio que nos pueda proporcionar información sobre el mercado de los juegos de mesa, pudiendo ir el contenido de dichos documentos desde valoraciones de juegos de mesa a comentarios en diferentes redes. Para ello, consultaremos diferentes fuentes, estudiaremos las fortalezas de cada una y escogeremos aquellas que sean más adecuadas para nuestros objetivos, teniéndose en cuenta, por supuesto, la accesibilidad a las mismas. A continuación, aplicaremos algunas de las herramientas mencionadas a lo largo del capítulo 2, principalmente funciones del paquete `tidyverse`, para obtener un conjunto de datos ya preparado para los siguientes pasos.

Una vez tengamos preparada una estructura con la que desarrollar el trabajo, estudiaremos descriptiva y gráficamente el contenido de la misma, de forma tal que ya podamos empezar a extraer conclusiones sobre el mercado, tales como tendencias o valores numéricos que resuman comportamientos. Además, buscamos ser capaces de detectar patrones al hacer esto, así como encontrar cualquier error que pudiésemos haber cometido a la hora de crear el conjunto de datos, pasando de esta forma nuevamente al punto anterior. Ahora, disponiendo de un objeto ordenado y sobre el que ya tenemos múltiples hipótesis, buscamos poder contrastarlas, apoyándonos en varios de los modelos estudiados a lo largo de la carrera, principalmente aplicados a la predicción de la puntuación de un juego de mesa, la que será nuestra variable objetivo, que representará la importancia del juego dentro del mercado.

Complementaremos este trabajo con la aplicación de diversas herramientas de la minería de texto, y más concretamente del procesamiento del lenguaje natural, con el objetivo de ser capaces de resumir el contenido de comentarios en un único valor numérico, o de poder construir una jerarquía entre juegos basada en la similitud de sus comentarios. Estas estrategias, no vistas a lo largo del grado, son nuevas para nosotros

y por lo tanto las destacamos como un objetivo en sí mismo: el aprender cómo emplear algunas de las estrategias clásicas de estos campos para tratar, de forma automática, grandes conjuntos de documentos de texto, buscando condensar la información contenida en ellos y/o diseñar variables auxiliares para la construcción de modelos de regresión o de clasificación.

Por último, buscaremos contrastar las hipótesis planteadas a lo largo del proyecto, siempre con el objetivo de obtener información sobre temáticas clave, puntos principales a tener en cuenta y, en general, cuáles son las variables que han ser controladas para garantizar el mayor éxito posible en un juego de nueva producción.

1.4 Estructura del documento

Este proyecto se encuentra dividido en tres bloques de contenido muy diferenciados, conteniendo el primer bloque, de introducción y preliminares, los capítulos 1 y 2; el segundo bloque, el núcleo del trabajo, los capítulos 3, 4, 5 y 6; y, por último, el tercer bloque, de conclusiones y posibles extensiones del proyecto.

El primer bloque nos servirá para conocer, en líneas generales, los objetivos del trabajo y las herramientas que emplearemos para lograrlos, desglosando no solo el *software* que nos será necesario, sino también las estructuras y modelos matemáticos en los que nos apoyaremos. Se incluye en el capítulo 2, adicionalmente, una breve descripción de los diferentes paquetes de R a los que recurriremos.

El segundo bloque contiene a los capítulos sobre extracción y lectura de datos, análisis descriptivo y gráfico, procesamiento del lenguaje natural y modelización, y supondrá el punto central del proyecto, en el que intentaremos explotar la información que obtendremos de la plataforma *boardgamegeek.com*, aplicándole las herramientas y los procedimientos que ya se explicaron en el primer apartado, buscando responder con esto a diferentes preguntas que iremos planteando a lo largo del trabajo.

Por último, en el tercer bloque comentaremos principalmente las conclusiones extraídas de este proyecto, complementadas, entre otros, con los posibles caminos que podrían complementar al ejercicio aquí desarrollado, y que servirían de extensión de los resultados obtenidos.

1.5 Sobre el estilo

En lo referido a la forma en la que escribiremos, destacamos aquí los convenios principales, que aplicaremos en todo momento a lo largo del proyecto:

- Los nombres de funciones, de carpetas, de objetos de R, de variables y palabras en otros idiomas serán todos escritos en cursiva. Ejemplo: “la variable *averageweight* resulta clave en este análisis”. Adicionalmente, los nombres de los juegos de mesa se escribirán también en cursiva, para destacarlos.
- Los nombres de los diferentes paquetes de R con los que trabaremos aparecerán en la fuente monoespaciada Courier New. Ejemplo: “nos apoyaremos en el paquete `tidytext` para llevar a cabo este estudio”.

2 PRELIMINARES

En este capítulo hablaremos tanto de las herramientas digitales empleadas a la hora de realizar el trabajo como del fundamento de las mismas, esto es, el marco teórico en el que se ubican, si bien no profundizaremos demasiado al ser este trabajo de enfoque esencialmente práctico.

2.1 Software

Dado que planeamos trabajar con grandes conjuntos de datos, se hace imperativo el uso de una herramienta que sea compatible con multitud de funciones, desde lo más básico hasta poder llevar a cabo, con relativa rapidez, un análisis del sentimiento o emplear técnicas de *machine learning*, por ejemplo. Además, un factor a valorar a la hora de elegir un lenguaje de programación es lo familiarizados que podamos estar con él.

Dado que a lo largo de estos años en el itinerario universitario hemos estado empleando de forma sistemática el lenguaje R y el entorno de desarrollo integrado RStudio, parecen las opciones lógicas para llevar a cabo nuestra labor, pues verifican todas las condiciones ya mencionadas.

Una de las principales razones para trabajar con R, sin embargo, es que además de ser uno de los lenguajes de programación más importantes hoy en día, y el más relevante en el ámbito estadístico [35], dispone de una comunidad muy activa, que no solo ofrece su ayuda en diversos foros, sino que además crea paquetes de toda índole, como los que mencionaremos en los siguientes subcapítulos.

2.2 Paquetes de R

Como ya se ha comentado, en R disponemos de multitud de paquetes, es decir, colecciones ya construidas que extienden el contenido básico de R, que nos pueden ayudar en nuestra labor, agilizando multitud de pasos al estar estos ya implementados como funciones o conjuntos de datos pregenerados.

Destacamos en la siguiente relación los paquetes más importantes empleados a lo largo del trabajo.

2.2.1 El paquete `tidyverse`

El paquete `tidyverse` es un conjunto de varios paquetes que, si bien de diferente funcionalidad y aplicación, tal y como indica su página oficial [14], comparten una

filosofía común en lo referido a la programación en R, recurriendo para ello a estructuras comunes, patrones de nominación y gramaticales semejantes y, ante todo, hay que remarcar que están diseñados para funcionar correctamente unos con otros.

Dentro de los paquetes que se incluyen en `tidyverse` los más relevantes para este trabajo son `dplyr`, `tidyr`, `tibble` y `ggplot2`, complementados con otros paquetes como `readr` o `forcats`, que empleamos pero en menor medida a lo largo del proyecto.

En primer lugar nos encontramos con el paquete `dplyr`, utilizado principalmente para transformar datos mediante diversas operaciones, yendo estas desde la reordenación por nombres hasta la agrupación y posterior modificación de variables. Esta herramienta resultará vital en cada paso que demos, pues no hay ocasión en la que los datos con los que trabajemos se encuentren en un formato adecuado para todo lo que queramos hacer, viéndonos, de esta forma, obligados a transformar los datos para ajustarlos a nuestras necesidades. Para más información sobre el paquete `dplyr`, se recomienda consultar el manual de referencia del paquete, enlazado en [10].

Adicionalmente, es relevante destacar que adoptaremos el estilo sugerido por los creadores de este ecosistema de paquetes a la hora de ir encadenando secuencias de operaciones empleando tuberías (incorporadas en `dplyr`, y tomadas originalmente de `magrittr`).

En segundo lugar, parafraseando a Hadley Wickham en el capítulo 12 de [20], "*todos los conjuntos ordenados se asemejan, pero cada conjunto desordenado lo es a su manera*"; esto es, si bien a la hora de aplicar herramientas estadísticas utilizaremos casi siempre conjuntos de datos de estructuras semejantes y ordenados, cuando nos enfrentamos a un nuevo problema, con datos aún desordenados, nada nos garantiza que dicho desorden no sea uno completamente nuevo para el usuario, diferente a cada conjunto previo con el que ha podido trabajar, a pesar de que el objetivo que tenga en mente, el conjunto de datos ordenado, sea "el mismo de siempre". Es por esta razón que es vital disponer de instrumentos que nos permitan reestructurar conjuntos de datos, tal y como veremos más adelante, especialmente en la sección 3.2. Concretamente, con `tidyr` obtenemos un paquete que puede ayudarnos a lograr este objetivo, proporcionándonos el paquete utilidades para intercambiar filas por columnas, desanidar listas y, en general, obtener un conjunto de datos ordenado, que tal y como se indica en [20] es aquel en el que cada variable tiene una columna, cada observación una fila y tal que cada valor se encuentra en una única celda. Para más información acudir al manual oficial, enlazado en [13].

Siguiendo ahora con la idea de trabajar con conjuntos de datos ordenados, el paquete `tibble` nos proporciona una estructura que intenta ser el siguiente paso adelante desde los conocidos *data frames* de R. Entre otras funcionalidades, el recurrir a *tibbles* y no a estructuras *data frames* nos permite obtener una configuración con mayor libertad a la hora de declarar los nombres, suprimiendo algunos de los comportamientos por defecto de los conjuntos *data frame* y, ante todo, nos otorga un modelo general sobre el que trabajar con el resto de paquetes de `tidyverse`, pues todos están diseñados con la idea de emplear tibbles.

Por último, dentro de los paquetes que más hemos empleado a lo largo del trabajo, el paquete `ggplot2` merece una mención especial, pues es una de las herramientas más potentes de las que dispone un estadístico hoy en día a la hora de comunicar resultados. Fundamentado en *The Grammar of Graphics* (véase [21]), ggplot2 nos proporciona no solo formas de hacer representaciones gráficas, sino que transmite también una filosofía, una gramática, tal y como nos indica Wickham en la introducción de [19], en la cual se considera cada posible gráfico como la unión de diferentes componentes igualmente importantes. De esta forma, se trabaja con varias capas que se van aplicando una tras otras, empezando con los propios datos y pasando por el escalado y las geometrías entre otros, siendo el gráfico el resultado de ejecutarlas todas a la vez. Cabe destacar que la sintaxis de `ggplot2` difiere considerablemente de la que uno podría esperarse tras haber empleado las funciones de representación básicas de R, como *hist* o *plot*, pues estas no comparten la ideología expuesta. Para más información sobre las utilidades del paquete, acudir al manual enlazado en [11]; si bien para proceder paso a paso, de forma más asistida, se recomienda el ya mencionado libro de Wickham, referenciado en [19].

En resumen, `tidyverse` se revela como un conjunto de paquetes de gran utilidad para cualquier proyecto y, en particular, en este trabajo. Es por ello que, dado que se llevará a cabo un uso extensivo de las herramientas incluidas en el paquete, intentaremos ser tan fieles como nos sea posible a la filosofía del paquete `tidyverse`.

2.2.2 El paquete `tidymodels`

Primeramente, hay que aclarar que aunque tenga un nombre semejante al del conjunto de paquetes ya visto `tidyverse`, el alcance de `tidymodels` es ligeramente diferente, siendo nuevamente una colección de paquetes, pero en esta ocasión orientados a la construcción de modelos estadísticos y de *machine learning.*

Dentro de `tidymodels`, tal y como nos indica su página oficial [12], nos podemos encontrar con multitud de paquetes que sirven a una u otra función dentro del sector de la modelización: `parsnip`, para implementar modelos; `recipes`, para modificar la definición de las matrices de diseño y en general establecer los pasos necesarios de preprocesamiento de los conjuntos de datos implicados en un *pipeline* o *workflow* de trabajo para la creación y evaluación de modelos; `tune`, para controlar los hiperparámetros de nuestros modelos; *`yardstick`*, para obtener métricas de nuestros resultados; entre otros. Es por estas razones que `tidymodels` es un paquete utilizado con gran frecuencia en el ámbito estadístico, pues aglutina un gran número de útiles vitales en la construcción y evaluación de modelos dentro del sector de la ciencia del dato.

Otros paquetes como `caret` o `mlr` tenían un enfoque integrador similar, pero en el caso de `tidymodels`, sucesor de `caret` en el que han estado involucrados desarrolladores de este junto con miembros del equipo de `tidyverse`, se ha pretendido refactorizar el diseño y hacerlo más compatible con las estructuras y el enfoque de `tidyverse`, dado su incuestionable éxito a lo largo del último lustro.

A lo largo del trabajo intentaremos ajustar los datos generados a diferentes modelos con el objetivo de obtener predictores, e intentaremos regirnos por las recomendaciones que hacen Kuhn y Silge en su libro sobre modelado en R [3].

2.2.3 El paquete `tm`

El paquete `tm` **—** siglas para *text mining* **—,** incluye multitud de funciones que agilizan cualquier tarea que tenga relación con la explotación estadística de grandes conjuntos de datos de texto, siendo especialmente útil a la hora de construir matrices de término-documento y de realizar transformaciones sobre los conjuntos de datos, tales como eliminar signos de puntuación, pasar a minúscula o eliminar las *stopwords*. Estas herramientas serán útiles a lo largo de todo el trabajo y especialmente cuando abordemos el modelo vectorial, si bien es cierto que nos obliga a trabajar con los datos organizados de una forma muy concreta, tal y como se explicará en detalle en el punto 2.5.2, en el que hablaremos sobre el modelo ya comentado. Aunque podamos encontrar un desglose completo de todas las funcionalidades que incluye el paquete en el manual de referencia [15], en la práctica no necesitaremos tal cantidad de utilidades, y nos bastará con seguir las pautas que da Ingo Feinerer en *Introduction to the tm Package* [2], documento en el que hace un desglose del flujo de trabajo habitual con el paquete `tm`.

2.2.4 El paquete `tidytext`

Tal y como ocurre con `tidyverse`, `tidytext` intenta implementar la filosofía del trabajo orientado a los conjuntos de datos ordenados, en los que, como ya mencionamos, cada variable es una columna, cada observación una fila y cada unidad observacional una celda de la tabla. Este paquete nos interesa en especial medida dado que los autores del mismo explican, paso a paso, cómo ejecutar un análisis del sentimiento aprovechando toda la potencia de sus funciones, encontrándose este proceso documentado el capítulo 2 de *Text Mining with R* [18], la que será nuestra referencia a la hora de aplicar las herramientas estadísticas más potentes de las que disponemos, orientadas al procesamiento del lenguaje natural.

Para ubicar el uso del paquete `tidytext` dentro del flujo de trabajo habitual de un análisis de texto diseñado con la filosofía dada por `tidyverse`, representamos aquí la siguiente figura, en la que queda clara la utilidad de cada una de las diferentes herramientas que hemos expuesto en los últimos puntos:

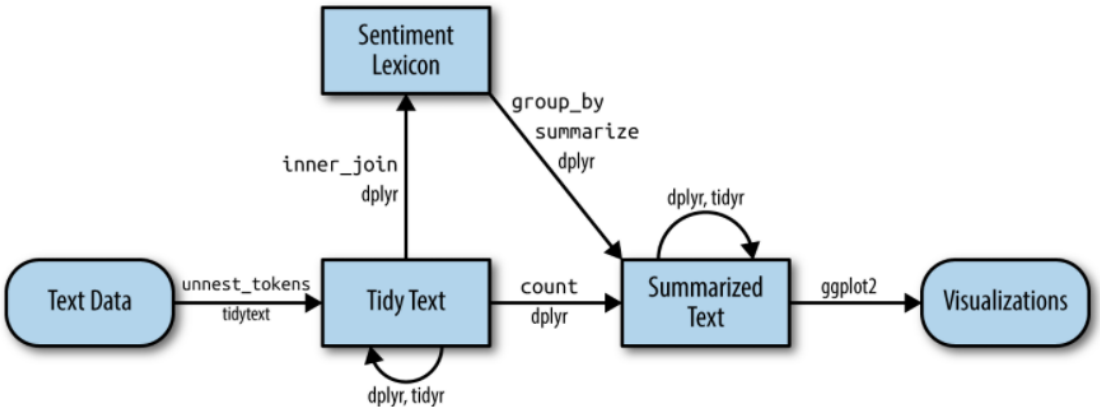

Figura 2.1: Flujo de trabajo y `tidytext`

Fuente: Figura 2.1 de [18]

Tal y como podemos observar, para obtener un conjunto de datos ordenados aplicamos primero *unnest_tokens*, función básica de `tidytext` que nos permite dividir fragmentos de texto en varias unidades, ya sean estas palabras o frases, tal y como se verá en las secciones 2.5 y 5.2; pudiendo combinar esta operación con otras propias del paquete `dplyr` para poder procesar correctamente los datos ya mencionados.

2.2.5 El paquete `wordcloud`

"*Una representación puede proporcionar respuestas a preguntas que no sabías que tenías*",

Ben Schneiderman, catedrático de la Universidad de Maryland

El paquete `wordcloud` es una de las principales herramientas que todo analista de texto que recurra a R conoce, pues a pesar de no proporcionar resultados numéricos o respuestas contrastadas a preguntas que uno pueda plantearse, nos otorga un conciso resumen, de rápida elaboración, de los datos con los que trabajamos, siendo por lo tanto un recurso más que estimable. De esta forma, y siguiendo el espíritu de la cita de arriba, este paquete nos da la oportunidad de no solo contrastar de forma inmediata determinadas cuestiones a las que podríamos hacer frente, sino que también nos puede dar ideas sobre otras preguntas que nos podrían interesar.

Realmente se trata de un paquete relativamente pequeño, con pocas funciones, pero de gran utilidad todas. Concretamente, nosotros nos centraremos en el uso de dos de las funciones principales, *comparison.cloud* y la función principal, la homónima *wordcloud*. Estas nos permitirán representar la información contenida en documentos de una forma concisa y rápida, representando las palabras con un color y un tamaño variable en función de la frecuencia absoluta de las mismas, gráficos que nos será especialmente de ayuda en el apartado 2.6, en el que trataremos el análisis de sentimiento. Además, *comparison.cloud* nos permitirá hacer la misma representación, pero agrupando las palabras según una categoría preseleccionada, como podría ser la fuente o el sentimiento asociado, como veremos más adelante.

Para más información sobre este paquete y sus otras funcionalidades acudir a la referencia [16].

2.2.6 Otros paquetes

Aunque no sean el núcleo de este trabajo, destacamos el uso de otros paquetes que nos han resultado de ayuda, tales como `magrittr` o `lubridate`, pues nos han permitido compactar el código y restringirnos a una sintaxis de tuberías en el caso del primero; y nos han ayudado en el momento de estimar el tiempo de ejecución de las partes más pesadas, computacionalmente, del código, cuestión especialmente relevante al trabajar con conjuntos de datos de tamaño considerable.

Otro paquete a destacar es `xml2`, pues debido a la plataforma que hemos escogido como principal fuente de datos, sus funciones de lectura y tratamiento de documentos en formato XML han sido vitales para el correcto desarrollo del trabajo.

2.3 Acceso a APIs

En multitud de ocasiones hemos oído a alguien hablar sobre el acceso a la API de una u otra plataforma web; sin embargo, ¿qué son exactamente? Incluimos este punto al jugar más adelante un papel fundamental en el capítulo 3, sobre el procesamiento de los datos.

Guiándonos por la explicación que da Brian Cooksey en su introducción a las APIs [1] una API (*application programming interface*) no es más que "una herramienta que simplifica y agiliza el acceso a los datos de una página web desde el ordenador de un usuario que busque leer, escribir o editar información contenida en la misma".

En la práctica, una API no es más que una funcionalidad más que se ejecuta junto con otras utilidades del servidor, dándole acceso al usuario a datos y a procedimientos que, de otra forma, tardaría horas en obtener. Sin embargo, habitualmente el uso y aprovechamiento de las APIs se encuentran separados en dos partes: la parte del cliente y la del servidor. La primera es la que solicita los resultados de nuestra consulta y los procesa una vez recibidos, como cuando hacemos una búsqueda con el móvil, cuyos resultados son generalmente interpretables; mientras que la parte del servidor ejecuta la consulta como tal, con todo el procesamiento y acceso de datos que esto le suponga, y devuelve el resultado a la parte del cliente, dejándole a esta la labor de adaptar los datos recibidos para que el usuario pueda entenderlos.

En la sección 3.1 veremos que el núcleo de la información más extensa que hemos podido encontrar en el ámbito de los juegos de mesa nos la ha proporcionado una API, a la que hemos tenido que requerir intensivamente datos sobre los juegos analizados.

2.4 Minería de texto

Ya mencionamos que el libro de Silge y Robinson, *Text Mining with R* [18] explicaba cómo trabajar con grandes conjuntos de datos de tipo texto. Sin embargo, para saber con precisión qué es lo que pretendemos hacer en los siguientes capítulos, es necesario explicar, aunque sea de forma breve, qué es la minería de texto, cuestión por la cual incluimos esta sección en el proyecto.

2.4.1 ¿Qué es la minería de texto? ¿Cómo se emplea?

Tal y como indica Ted Kwartler en su libro sobre la minería de texto en la práctica [4], la minería de texto es el "*proceso mediante el cual extraemos información útil de textos*", con el objetivo, generalmente, de resumir lo contenido en grandes agrupaciones de texto. Sin embargo, además de un campo teórico dentro de la estadística y las ciencias de la computación, en la práctica sirve a una función muy diferente, usándose por lo general dentro de múltiples empresas como ayuda a la hora de tomar decisiones, tal y como veremos en el siguiente punto.

De esta forma, la minería de texto abarca un amplio abanico de procedimientos y de funciones que sirven para explotar la información contenida en grandes conjuntos de datos de texto, generalmente proporcionándonos valores numéricos que logren resumir la información contenida en los documentos, como el número de apariciones por palabra o el "sentimiento" asociado a cada palabra o a cada documento, tal y como veremos en la sección 2.6.

2.4.2 Importancia de la minería de texto

Hoy en día se generan ingentes cantidades de datos cada segundo, y si bien una importante proporción se encuentra en formatos más sencillos, ciertamente otra parte importante se almacena como cadenas de texto, de difícil interpretación en lo que a explotación estadística se refiere. Así, en plataformas como Twitter, Reddit o, en general, cualquier red social, aunque dispongamos de otros datos, el grueso de la información que podemos obtener de la página en cuestión estará formado por un conjunto de documentos que, si bien contendrán valores numéricos o categóricos como el usuario o el número de *likes*, a la hora de extraer información útil de estos resultados nos obligarán a disponer de herramientas que sirvan de ayuda a la hora de tomar decisiones, especialmente si estas son consideradas dentro del sector empresarial.

De esta forma, con la minería de texto es posible extraer conclusiones a partir de grandes corpus de texto, tales como la opinión pública sobre un tema, la popularidad de ciertos eventos actuales o la relevancia incluso de un comentario o de una persona dentro de una red, pudiendo predecir de esta forma comportamientos futuros dentro de la plataforma a partir de estímulos semejantes.

Al tiempo que multitud de empresas pasan de un comportamiento "industrial" a uno puramente informático, también la visión a largo plazo de las mismas reconoce lo imperativo de tener en cuenta los comentarios y la reacción del público, llegando a un punto en el que las empresas pasan a centrarse completamente en el cliente,

especialmente las que hoy en día son más exitosas, siendo ejemplo claros Google o Amazon. Siguiendo esta idea, el poder interpretar la respuesta de los usuarios es un punto vital a la hora de mejorar como empresa, siendo de esta forma necesario el recurrir a herramientas como la minería de texto.

Por comentar otros ámbitos y hechos que le dan valor a este ámbito, tal y como dice Kwartler en [4], se pueden mencionar los siguientes puntos:

- El contenido online sobre el mercado crece cada día, incluyendo el comportamiento de empresas rivales y de fuentes de recursos ajenas.
- La digitalización de registros oficiales se lleva a cabo poco a poco en muchos países, siendo un ejemplo claro la del sector de la seguridad social. Este tipo de documentación es especialmente adecuada para este tipo de análisis y procesamiento automático, teniendo en cuenta además las dimensiones de los datos con las que se puede trabajar.
- Nuevas tecnologías como la transcripción del lenguaje hablado nos permiten generar enormes cantidades de datos en formato de texto.

2.5 Procesamiento del lenguaje natural

El procesamiento del lenguaje natural, técnicamente una sección dentro de la minería de texto, merece una sección independiente y una atención especial al ser uno de los puntos más importantes en el proyecto, no solo por el interés en sí mismo sino porque también se trata de una gran novedad dentro de lo estudiado a lo largo del currículo académico, distanciándose considerablemente de otras ideas y procedimientos vistos a lo largo del grado.

El procesamiento del lenguaje natural es, como se explica en [6], una disciplina dentro de la Inteligencia Artificial centrada en el planteamiento y la investigación de métodos computacionales que permitan la comunicación entre personas y máquinas mediante los llamados “lenguajes naturales”, es decir, aquellos que utilizamos en nuestro día a día comunicándonos unos con otros. De esta forma, tal y como indica el nombre, las técnicas de este campo intentan, en un primer momento, obtener información concisa a partir de grandes conjuntos de documentos de texto. Por lo tanto, algunas aplicaciones de estos procedimientos son el comprender el funcionamiento del lenguaje en el sentido del **significado** subyacente; el recuperar información, como en un buscador de una plataforma web; o bien incluso traducir de forma automática, predecir el discurso o reconocer un mensaje a partir de la voz del usuario.

Dentro de este ámbito nos encontramos con multitud de herramientas que sirven al mismo propósito: interpretar o al menos resumir la información contenida en grandes conjuntos de datos de texto. Sin embargo, por cuestiones de extensión y al ser algo completamente nuevo para nosotros, nos restringiremos a trabajar con un par de técnicas, mencionando en todo caso posibles extensiones de las mismas.

Destacamos dentro del campo los modelos centrados en el contenido semántico definidos como en [6], que son aquellos que se centran en la obtención de información léxica a partir del corpus de trabajo, requiriéndose para ello de un gran número de documentos en el corpus. En estos casos, dado dicho corpus y una pregunta del usuario el algoritmo intenta, generalmente, obtener un conjunto de documentos como resultado y una representación de los mismos, tales como los documentos más semejantes a uno dado o una nube de palabras comparativa entre diferentes documentos. Nótese la importancia de la definición de "documento" en este modelo, pues hay que diferenciar el caso en el que trabajemos con documentos que sean comentarios de una red social, por ejemplo, del caso en el que sean libros enteros. En nuestro caso, nos centraremos principalmente en una muestra de comentarios escritos por los usuarios de la plataforma *boardgamegeek.com*, tomándolos como opiniones de los juegos incluidos en dicho portal.

En definitiva, con el procesamiento del lenguaje natural intentamos crear herramientas que nos permitan entender el lenguaje en el que hablamos normalmente, pudiendo emplear esta comprensión para dar posteriores pasos con los datos ya resumidos y organizados.

2.5.1 El modelo de unigramas

El modelo de unigramas, realmente un conjunto de modelos centrados en el contenido semántico, categoriza a todo aquel modelo que se fundamente en contar el número de apariciones de cada palabra, probablemente en cada documento. De esta forma, en esta categoría podemos catalogar tanto el modelo de espacio vectorial como el análisis de sentimiento que plantearemos en 5.1.1, pues el fundamento de los dos es semejante al basarse en contar las apariciones de las palabras de interés, si bien luego asignaremos pesos muy diferentes. De esta forma, podemos considerar el modelo de unigramas como un grupo de procedimientos iniciales que nos permiten iniciar nuestro análisis del texto, si bien luego comentaremos algunos de los problemas que padecen estos modelos, en general.

2.5.2 El modelo de espacio vectorial

Para aplicar este modelo, basado en unigramas, es primeramente necesario considerar un vocabulario relevante para el usuario, esto es, un conjunto de palabras importantes para la consulta realizada. Con este modelo buscamos definir la relevancia de un documento dada dicha consulta, de forma tal que se devuelva, por ejemplo, el documento más relevante dada una búsqueda. Alternativamente, también se puede emplear para medir la semejanza entre varios documentos independientes, obteniendo de esta forma también cierta jerarquía que puede resultar de utilidad.

Cabe destacar que, si bien un punto fundamental del modelo es la consideración de las "frecuencias", que ahora definiremos, habremos de darle menor peso a aquellos términos que aparezcan muchas veces en multitud de documentos, pues no aportan demasiada información en un primer momento. Para justificar esta decisión, tomemos como ejemplo el que aplicaremos más adelante, la comparación entre comentarios de un juego de mesa. Aunque la palabra "juego" se encuentre en nuestro vocabulario, al aparecer en prácticamente cada comentario, realmente no nos dice nada sobre la semejanza entre diferentes documentos.

Yendo ahora a la implementación del modelo, para obtener la representación vectorial de un documento, que al final es nuestro objetivo, necesitamos calcular primeramente diferentes elementos, siguiendo las indicaciones de [6]:

- La frecuencia de un término en un documento, $tf_{t,D}$, que mide las apariciones de un término t en un documento D dado.
- La frecuencia documental, dt_t, que mide el número de documentos en los que aparece un término.
- La frecuencia documental inversa, $idf_t = log\left(\frac{N}{df_t}\right)$, donde N es el número total de documentos.
- El peso de un término t en un documento D, $tfidf_{t,D} = tf_{t,D} \cdot idf_t$

Ahora bien, dado un vocabulario $V = \{t_1, \dots, t_r\}$ se define la representación vectorial de un documento D como: $\overline{D} = (d_1, \dots, d_r)$, donde $d_i = tfidf_{t_i,D}$.

Conocida ahora esta representación es posible medir la semejanza entre diferentes documentos, definiéndose la similitud entre dos documentos como:

$$sim\left(\overline{D}, \overline{D'}\right) = \frac{\sum_{i=1}^{r} d_i d_i'}{\sqrt{\sum_{i=1}^{r} d_i^2}\sqrt{\sum_{i=1}^{r} d_i'^2}}$$

Este modelo, conocido como modelo de Grossman, es de fácil implementación, y proporciona a pesar de ello resultados prometedores, si bien tiene multitud de inconvenientes para según qué estudios, tal y como mencionaremos a continuación.

2.5.3 Limitaciones de los modelos de unigramas

Hasta ahora podemos considerar que los modelos aquí planteados pueden proporcionar resultados más o menos buenos, dependiendo del caso. Sin embargo, al trabajar con modelos basados en unigramas, y concretamente el modelo vectorial, nos podemos encontrar con varios problemas, que desarrollaremos en este punto.

Primeramente, uno podría considerar la dificultad de interpretar, con estos modelos, la negación, pues si bien podemos contar el número de apariciones de partículas negativas como “no”, “ni” o “nada”, estas pueden usarse de varias formas, cambiando completamente el significado de la frase sin cambiar las palabras que aparezcan en ella, lo que rompe cualquier idea previa de la que pudiese partir nuestro modelo. Por poner algunos ejemplos, las siguientes frases serían las mismas para los modelos basados en unigramas:

- *No está mal, me gusta*
- *Está mal, no me gusta*.

Con estos ejemplos, también hemos visto, de forma implícita, otro de los problemas de estos modelos, y es que no tienen en cuenta el orden, pudiéndose dar casos como los arriba expuestos, en los que una variación en el orden de aparición de las palabras modifica completamente el significado de la frase. Para evitar este factor es importante tener en cuenta qué palabras aparecen generalmente después de unas dadas, trabajando con lo que denominamos la correlación entre términos, si bien esto no llegaremos a tratarlo en este proyecto al estar fuera del alcance del trabajo planteado.

Otro punto a tener en cuenta es que una mala elección del vocabulario, en aquellos supuestos en los que sea necesario, como en el modelo vectorial, puede proporcionar poca información sobre la realidad que estamos estudiando, o podría incluso tergiversarla, creando semejanzas y patrones que, si bien podrían existir, no son aquellos que generalmente serían de utilidad para un analista de datos, tales como el obtener una gran similitud entre diferentes documentos debido únicamente a que ambos están escritos con un registro culto, o cometiendo las mismas faltas de ortografía. Estos son casos que, si bien es fácil darse cuenta de que se dan, no dejan de dificultar la elección del vocabulario, ya compleja por sí misma.

2.5.4 El modelo de ratios

Intentando ir más allá, podríamos considerar como unidades del lenguaje (*tokens*) frases en lugar de palabras individuales, y trabajar sobre ellas para obtener información más completa que la que nos proporcionan los modelos de unigramas clásicos. Por lo tanto, nos planteamos, dada una separación de un conjunto de documentos en frases, el modelo de ratios, que es aquel que le asigna a cada una de dichas frases una fracción, a saber, el cociente entre el número de palabras verificando cierta propiedad dentro de una frase y el número de palabras en la misma.

Así, por ejemplo, considerando la propiedad "pertenecer al lenguaje culto" y el siguiente documento:

> "*Juego ágil y de fácil comprensión. Recomendado para todos.*"

Primeramente separaríamos en unidades:

- T1: "*Juego ágil y de fácil comprensión*".
- T2: "*Recomendado para todos*".

Ahora, suponiendo que "ágil", "comprensión" y "recomendado" están catalogadas como palabras cultas, obtendríamos las siguientes ratios, teniendo en cuenta que en T1 hay 6 palabras y en T2 solo 3:

- Ratio(T1): 2 / 6 = 1 / 3
- Ratio(T2): 1 / 3

Se puede tener en mente que quizás, quitando palabras de poca relevancia, como "y" o "de", la ratio de palabras cultas dentro de las que tienen una semántica fuerte asociada se incrementaría, lo que facilitaría la interpretación de los indicadores creados y les daría una mayor consistencia.

Finalmente, para obtener la ratio asociada a un documento concreto podríamos tomar la media, la media geométrica o la mediana como representación del comportamiento promedio del documento. Una vez obtengamos dicha ratio, podríamos realizar varios análisis, desde obtener una jerarquía dada por nuestro vocabulario a una cercanía entre diferentes documentos.

Notemos que en este caso, a pesar de trabajar con frases, al menos teóricamente, en la práctica estamos contando el número de apariciones como ocurría con los modelos basados en unigramas, si bien es cierto que estamos modificando los pesos en función de la frecuencia relativa de las mismas al número de palabras en la frase en la que se encuentran.

2.5.5 Otros modelos

Terminamos esta sección mencionando que, si bien los modelos de procesamiento del lenguaje natural aquí considerados pueden proporcionar buenos resultados, tal y como veremos en el capítulo 5, en la práctica sería conveniente recurrir a un motor de procesamiento de texto más potente, que tuviese en cuenta algunos de los puntos ya mencionados: correlación entre palabras, interpretación de la negación o detección del sarcasmo, por mencionar algunos.

Sin embargo, tal precisión está fuera del alcance planteado para este proyecto, primeramente por extensión y en segundo lugar por complejidad computacional, pues aunque los modelos aquí expuestos se fundamenten en un conjunto de cálculos relativamente sencillos, al trabajar con grandes agrupaciones de documentos, un ordenador "normal" puede llegar a tardar horas en ejecutar algunos de los pasos aquí comentados. Un ejemplo claro sería el de separar diferentes documentos en las unidades de texto (palabras, frases u otro), operación que, aunque pueda parecer sencilla, en la práctica puede llegar a consumir un tiempo importante.

En definitiva, es por esta razón que nos limitaremos a emplear los modelos expuestos en los últimos puntos, aun sabiendo que existen otros, como los probabilísticos, que pueden proporcionar mejores resultados, con un coste en tiempo mucho mayor, por lo general.

2.6 Análisis del sentimiento

Recordemos la Figura 2.1, en la que aparecía "*Sentiment Lexicon*" como paso intermedio dado un conjunto de textos ordenados, cuyo significado no llegamos a explicar. Hasta ahora, simplemente nos hemos restringido a un vocabulario prefijado, un conjunto de palabras que por una u otra razón resultan de interés, ya sea por conocer las menciones de un producto, el número de veces que una persona es felicitada por cierta cuestión o la relevancia de un evento en una red social; sin embargo, ¿podemos hacer más con una selección adecuada del vocabulario?

El análisis del sentimiento, campo incluido dentro del procesamiento del lenguaje natural, busca extender los primeros resultados que hemos expuesto, trabajando con un vocabulario muy específico, que nos permita obtener conclusiones en materia del "sentimiento" asociado a cada comentario.

Este campo hereda todas las ventajas y posibles problemas de los modelos estudiados hasta la fecha, pues al fin y al cabo necesita fundamentarse en alguna herramienta que

nos permita separar en diferentes unidades el texto, para una posterior asignación de sentimiento a cada una, como veremos a continuación.

2.6.1 Fundamentación

Cuando una persona lee un texto, nuestra comprensión de la intención del autor a la hora de transmitir una u otra sensación nos permite ir más allá del significado particular de cada palabra, obteniendo una interpretación global de cada fragmento, interpretación que podría catalogarse como una respuesta en nosotros, un sentimiento. Nuestro interés, por tanto, es ser capaces de obtener de forma algorítmica dicho sentimiento dado un texto. Como se indica en el capítulo 2 de [18], una forma de construir el sentimiento de un texto es considerándolo como la suma de los sentimientos de las diferentes unidades que lo componen.

A la hora de obtener dicho "sentimiento" individual, la primera opción que se nos podría ocurrir es hacer un recuento, por ejemplo, del número de palabras positivas que aparecen en cada documento del corpus. Además, sería importante destacar la existencia de partículas que modifican el significado o el peso de la siguiente palabra, como por ejemplo la palabra "algo", que podrían relativizar el peso de "lento", si bien para poder tener esto en cuenta deberíamos recurrir a técnicas más sofisticadas. Sin embargo, esto no tendría en cuenta el número de palabras que aparecen en dicho documento, ante lo cual podríamos, al igual que hicimos en la sección 2.5.4, aplicar un modelo de ratios, obteniendo una especie de frecuencia relativa de palabras positivas, que interpretaríamos como sentimiento.

Un problema que tiene este proceder es que no tiene en cuenta la aparición de palabras de otra categoría de sentimiento en el mismo texto. Así, por ejemplo, si se tratase de una crítica completa de un juego, mencionando sus puntos fuertes y débiles, probablemente apareciese un gran número de palabras negativas, pero aun así no por ello deberíamos catalogarla como un documento con un sentimiento negativo. Es por ello que conviene relativizar el número de palabras de un tipo de sentimiento concreto (positivo, negativo, alegría…) respecto de otro, ya sea considerando la diferencia entre sus apariciones en cada documento o tomando el cociente, modificado para no obtener nunca un cero en el denominador. Nótese que una palabra podría tener asociados varios sentimientos, y que dependiendo de la situación unos podrían anularse con otros a la hora de obtener el balance final de una frase o de un documento entero. Para tener este último factor en cuenta sería necesario profundizar en la construcción de la frase en la

que se encuentra, para lo cual deberíamos recurrir a correlaciones o a otros procedimientos más complejos.

Por último, otra forma de obtener el sentimiento en un texto, concretamente mediante métodos de carácter más numérico, es la que se basa en diccionarios (*lexicons*) como el AFINN [9], diseñado por Finn Årup Nielsen, profesor de la Universidad Técnica de Dinamarca, con el objetivo de realizar un análisis de sentimiento. En este caso, para cada palabra existen varias categorías a las que puede pertenecer o no – diferentes sentimientos –, y en cada caso, para cada combinación, nos encontramos con un peso positivo o negativo, una polaridad que varía por ejemplo, en el AFINN generalmente entre -5 y +5. De esta forma, tenemos en cuenta no solo el número de apariciones de palabras positivas o negativas, sino también el peso más probable que pudiera tener cada una dentro de una frase. Así, si queremos obtener el sentimiento de la frase:

"*Juego espectacular. Algo lento igualmente*",

los otros métodos comentados le darían tanto peso positivo a "espectacular" como peso negativo a "lento", obteniendo por ello un sentimiento nulo, a pesar de que en este caso parece claro que el sentimiento debería ser positivo, no dándole el mismo peso a las dos palabras ya mencionadas.

Resumiendo, dentro del análisis del sentimiento encontramos multitud de formas de proceder, todas basadas en la idea de que las palabras que componen un texto se encuentran cargadas de intención, de sentimiento, y por lo tanto es posible identificar el sentido que el autor intentaba darle al mismo, mejorando de esta forma nuestra comprensión del documento mencionado.

2.6.2 Aplicaciones

Tal y como mencionábamos hasta ahora, el análisis del sentimiento busca identificar una emoción subyacente al texto, generalmente explicitada a través de las palabras que lo componen. Por lo tanto, es lógico pensar que esta herramienta es de gran utilidad a la hora de, por ejemplo, saber cómo responden grandes grupos de personas a determinados estímulos, en las redes sociales, por ejemplo.

Siguiendo esta idea, enfocando las aplicaciones desde la mentalidad ya mencionada de las empresas centradas en el cliente, algunas podrían ser:

- Estudiar cómo diferentes grupos reaccionan ante los comentarios de un mismo partido político en una red social como Twitter.

- Obtener información adicional de los comentarios de una plataforma de venta, para contrastar la puntuación de la transacción con el sentimiento asociado al comentario del comprador.
- Identificar el interés y la respuesta del público ante eventos de interés como celebraciones, tragedias o noticias varias.
- Obtener información más allá de la que una plataforma pueda proporcionarnos en las especificaciones de un elemento almacenado con comentarios asociados, creando variables adicionales como “¿Asusta?” o “¿Alegra?”, como veremos en la sección 5.2.

En resumen, con el análisis de sentimiento podemos interpretar la respuesta de grandes grupos de gente a través de comentarios, obteniendo valores numéricos que logren resumir la forma en la que las personas receptoras han percibido un evento o equivalente de interés para el analista. En la sección 5.2 intentaremos extraer, concretamente, la información relativa a la percepción de los usuarios de los juegos de mesa a través de la plataforma *boardgamegeek.com*.

3 PROCESAMIENTO DE DATOS

Nuestro objetivo en este capítulo es buscar en un primer momento conjuntos de datos que puedan servir a nuestro propósito de analizar el mercado de los juegos de mesa, para posteriormente poder almacenar esos datos y trabajar con ellos, convirtiéndolos en datos ordenados a los que posteriormente podamos aplicar los modelos y procedimientos comentados en el capítulo anterior.

Dado que la totalidad de las fuentes consultadas se encuentra en inglés, y para intentar internacionalizar el proyecto en la medida de lo posible, recurriremos a una nomenclatura inglesa para todos los nombres de funciones, archivos y carpetas con los que trabajemos, facilitando de esa forma la tarea de comprender el código para los no hispanohablantes.

3.1 Fuentes de datos

Como ya mencionamos en la introducción, dada la relevancia del mercado de los juegos de mesa sería lógico suponer que podríamos disponer de multitud de bases de datos de las que partir a la hora de iniciar esta búsqueda de recursos. Si volvemos a [31], nos encontramos con que *boardgamegeek.com*, *cardmarket.com*, *games-workshop.com* y *edhrec.com* son las primeras páginas no dedicadas exclusivamente a implementar un juego en una plataforma online; sin embargo, de esas solo la primera nos puede ofrecer una visión global al ser el foro más conocido dedicado a los juegos de mesa, mientras que las otras pertenecen a sectores muy específicos dentro de los juegos de mesa, a saber: juegos de cartas coleccionables, *Warhammer* y otros *wargames* y *Magic the Gathering*. Concretamente, con una búsqueda descubrimos que *boardgamegeek.com* dispone de una API [24], y con ella es posible extraer multitud de datos de cada juego en su base de datos, así como comentarios asociados al mismo. Por lo tanto, tenemos una primera posible fuente de datos, la plataforma *boardgamegeek.com*.

Otras opciones que no aparece en la página citada y que se nos podrían ocurrir podrían ser las siguientes:

- Amazon: una buena fuente de información, especialmente si buscamos resultados de índole puramente económica. Sin embargo, no nos proporciona mucha información sobre los juegos de mesa como tal, pudiendo conocer únicamente la categoría y la descripción, entre otros pocos datos. Por otro lado, disponen de una API y de un tutorial de la misma que agilizan mucho el proceso de familiarizarse con la plataforma, pudiendo uno obtener sus primeros

resultados en poco tiempo. Cabe destacar también que el acceder a los comentarios podría ser una gran ayuda, ofreciéndonos contenido que tratar con los procedimientos vistos en el capítulo 2.

- Twitter: una fuente excelente de comentarios relativos a juegos mediante convenientes filtros, aunque debamos para ello trabajar con cantidades enormes de datos. Otro problema añadido es que hay que pedir un acceso a su API, y aunque sea fácil y gratis obtener el acceso estándar, este no da permisos como para acceder a publicaciones que no se hayan realizado en las últimas dos semanas. Para poder recorrer todo el archivo hace falta un acceso de tipo académico, y si bien puede ofrecerse gratuitamente para estudios y análisis, resulta difícil que se lo ofrezcan a un estudiante de grado, tal y como se puede ver en las opciones de solicitud de acceso.
- Kickstarter: una página de mecenazgo para proyectos de todo tipo, desde musicales hasta sociales, pasando, tal y como nos interesa, por los juegos de mesa. Aunque no disponga como tal de una API, el motor de búsqueda por defecto se puede modificar con gran facilidad para personalizar determinadas búsquedas, filtrando de esta forma resultados que nos puedan interesar.
- Páginas propias de editoriales de juegos de mesa como Devir, Nosolorol o Asmodee: podrían resultar interesantes, pero no nos proporcionan información alguna sobre la realidad económica de los juegos con los que trabajamos, más allá de su precio. Además, el no disponer de API en ninguno de dichos casos nos obligaría a ejecutar un *web scraping* para obtener la información de la que disponen, que como ocurría con Amazon tampoco es tanta como nos podría interesar, poco más que la descripción, el título y el precio.

A la vista de estos comentarios, y buscando el poder centrarnos en una fuente y explotar tanto como nos sea posible la información contenida en la misma, optamos por trabajar principalmente con *boardgamegeek.com*, explicando en las siguientes secciones el porqué de esta elección en detalle. Además, por comodidad, desde ahora nos referiremos a esta plataforma como BGG.

Buscamos ahora explicar de forma concreta qué es la BGG, qué información buscamos obtener de ella y de qué forma podremos hacerlo, explicando en el proceso el funcionamiento de los aspectos más importantes de su API, que tal y como ya mencionamos se encuentra desglosada en [24].

3.1.1 ¿Qué es BoardGameGeek?

Basándonos en la explicación que da la plataforma en sus preguntas más frecuentes [22], la BGG se autodefine como una “base de datos de juegos de mesa”, un recurso digital que se apoya en una comunidad activa para construir un catálogo con información sobre tanto juegos de mesa tradicionales como juegos de mesa modernos. Además, se declara como “el mayor y más actualizado lugar con información sobre juegos de mesa”, hecho que se corresponde completamente con los objetivos de nuestro proyecto. Entre estos datos nos encontramos con información sobre más de 300.000 juegos de mesa, incluyéndose en esta, entre otros, la duración, el año de publicación, la edad mínima o el número de jugadores mínimo y máximo.

Por otro lado, el sitio cuenta con más de 2 millones de usuarios registrados, que son los que votan en las diferentes encuestas para cada juego, generando de esta forma valores como el promedio de la puntuación o de la complejidad, así como el número de juegos recomendados o la clasificación entre diversos juegos por categorías, entre otros. Veamos un ejemplo de esta información en la página en la siguiente figura:

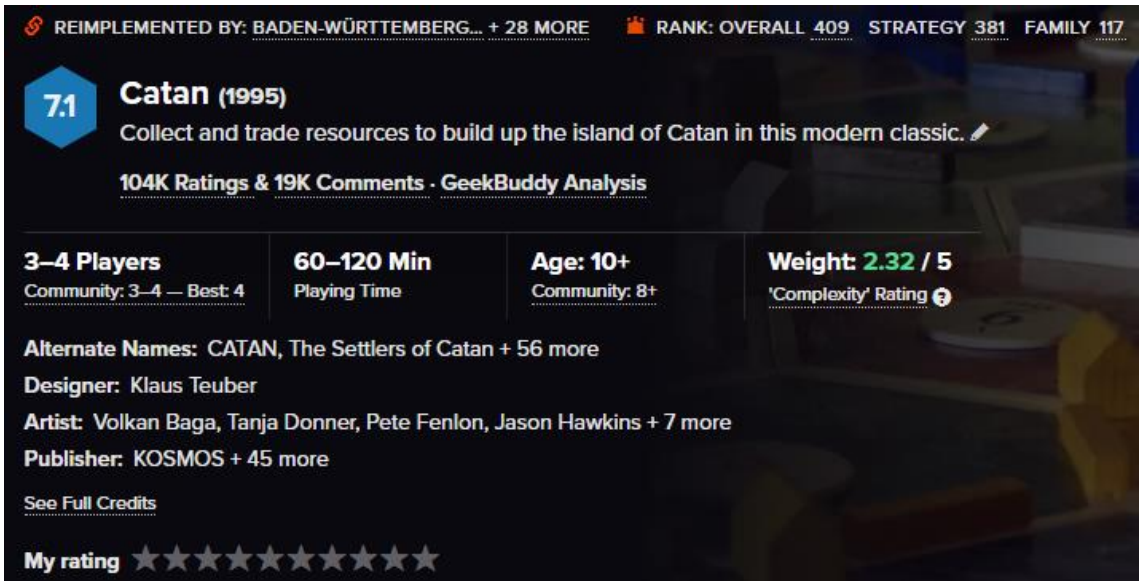

Figura 3.1: Información sobre un juego en *boardgamegeek.com*

Fuente: https://www.boardgamegeek.com/boardgame/13/catan

Finalmente, también incluye contenido relativo a las compras y ventas de juegos de mesa, si bien solo de forma “local”, esto es, entre miembros de la plataforma, permitiendo escoger entre “pide”, “quiere intercambiar” y “vende” como estados de un usuario frente a un juego. Estas variables, dado que podrían determinar en gran parte

el impacto del juego de mesa en el mercado o al menos explicarlos, habrán de ser estudiadas más adelante, incluyéndose en los modelos que construyamos.

Cabe destacar que la propia página establece que los datos en ella almacenados pueden ser explotados mediante fuentes externas a través de procedimientos de minería de texto, tal y como pretendemos hacer en este capítulo, y de hecho pone a disposición de los usuarios una API para llevar a cabo tal propósito.

Optamos por recurrir a esta plataforma frente a otras disponibles de la misma temática, como podrían ser *TableTopFinder, BoardGameFinder* o *BoardGameAtlas* al disponer la opción escogida de una API que facilita el manejo de los datos, pero principalmente por ser la plataforma más famosa dentro del ámbito, como ya comentamos. Adicionalmente, en estas otras no encontramos la información tan organizada y lista para su extracción como en la BGG.

Otro añadido, que hace de la BGG la opción más lógica, es que nos dota de multitud de tipos de datos, desde los comentarios de los usuarios en cada juego a las valoraciones, pasando por información propia del juego compactada y datos de la página, tipos que al fin y al cabo enriquecerán nuestro modelo y nos permitirán aplicar las herramientas vistas en el capítulo 2.

Resulta interesante saber que entre enero y junio de 2021 la página recibió más de 13 millones de visitas [33], siendo, a modo de comparativa, las visitas de la Universidad de Sevilla, *https://www.us.es/*, algo más de 8 millones en los mismos plazos [32].

3.1.2 Acceso a la API

Como ya hemos comentado, la BGG dispone de una API que nos facilitará el acceso a los datos que almacena la página. Sin embargo, como veremos más adelante, ni la ayuda de la API es demasiado completa ni la forma de almacenar los datos es la más correcta. Procedemos ahora a explicar el funcionamiento de la API, guiándonos por [24].

Primeramente, en la referencia comentan que están preparando una segunda versión de la API pero, dado que aún hay funcionalidades que no están implementadas en la BGG XML API2, pues tal es su nombre, optamos por recurrir a la BGG XML API. Notemos también que la segunda versión de la API se encuentra en beta, y que la primera dispone de un manual más claro.

El acceso a la BGG XML API no tiene otra interfaz más que la barra del navegador, esto es, el tipo de peticiones http que acepta es de tipo GET, luego podemos realizar consultas directamente introduciendo la url con los parámetros de búsqueda en el

navegador, o bien recurriendo al uso de software como POSTMAN. Así, por ejemplo, si escribimos:

https://www.boardgamegeek.com/xmlapi/search?search=Catan

obtenemos el siguiente resultado en formato XML que se muestra en la Figura 3.2:

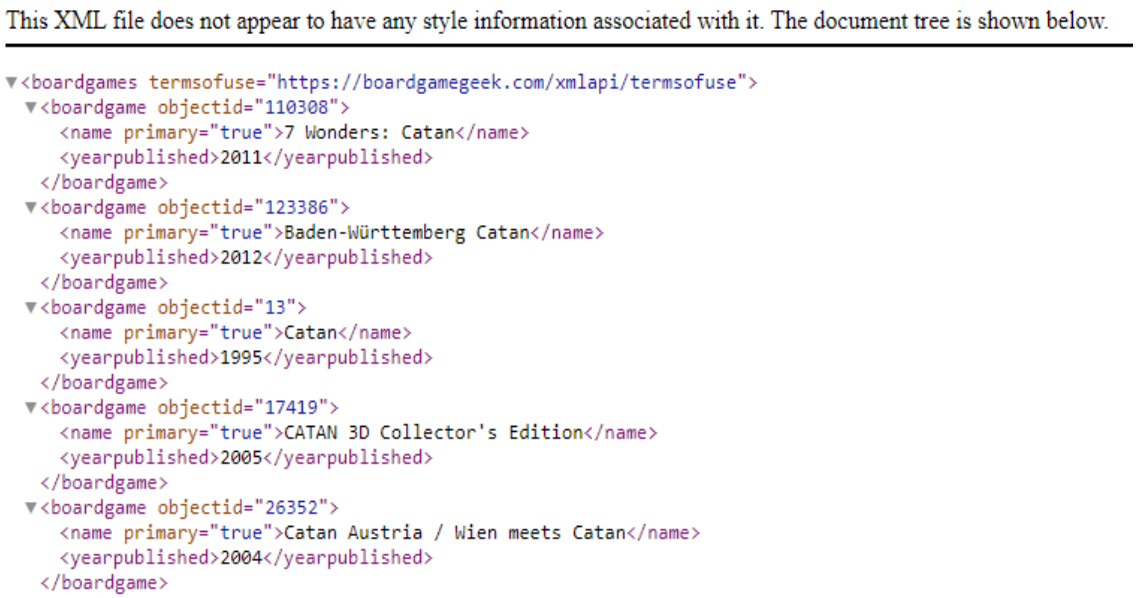

Figura 3.2: Ejemplo de consulta a la API de la BGG.

Fuente: Elaboración propia a partir de la API de la BGG.

Recordemos que XML es un acrónimo para *Extensible Markup Language*, el lenguaje de marcas autodescriptivo. Notemos que hemos instalado una extensión de Chrome para poder visualizar de manera más amigable los resultados, directamente en el navegador.

Por ahora hemos realizado una búsqueda, pero esta API dispone de multitud de opciones:

- Buscar juegos por nombre o por *aka* (*also known as*, a modo de alias) como ya hemos hecho.
- Obtener información sobre juegos particulares dadas sus ids, escribiendo después de “xmlapi/” las ids separadas por comas. En este caso incluye información más extensa que la de la búsqueda anterior, conteniendo, entre otros, nombres alternativos, información básica, encuestas sobre el número de jugadores recomendado, dependencia del lenguaje y edad recomendada, así como todas las relaciones con otros elementos tales como podcast, artistas o artículos.

- Extraer todos los comentarios sobre un juego seleccionado, estableciendo el atributo *comments* en 1, con un "?" primero, con la puntuación asociada a cada uno, si la hubiera. Por ejemplo: *http://www.boardgamegeek.com/xmlapi/boardgame/13?comments=1*
- Generar estadísticas mediante el atributo *stats* establecido en 1, de los puntos más importantes, pues se incluyen los valores: *average, averageweight, usersrated, owned, wanting, wishing* y la posición en diferentes clasificaciones. La primera es la media de las puntuaciones no nulas de los usuarios, mientras que la segunda es la media de las puntuaciones no nulas de la complejidad del juego.
- Restringir la información obtenida a un periodo particular, estableciendo el atributo *historical* en 1 y añadiendo los atributos *from* y *to* con una fecha en formato AAAA-MM-DD como valores de los mismos.
- Devolver los juegos en propiedad de un usuario seleccionado, así como sus intereses, sus valoraciones y, en general, casi cualquier lista que haya hecho en la plataforma. Dispone de varios tipos de filtros para seleccionar entre las diferentes "*geeklist*" del usuario. No le dedicamos más tiempo al no sernos de ayuda el interés de un usuario particular.
- Extraer los mensajes de un foro dentro de la plataforma. No nos centramos en este campo al ya tener los comentarios para cualquier juego que seleccionemos.

Notemos, primeramente, que no hay forma inmediata de extraer **todos** los juegos de mesa de la base de datos con rapidez, lo cual nos obligará a dedicarle un tratamiento especial en el capítulo 3.2. Además, los datos se encuentran en formato XML, y puede llegar a ser difícil trabajar con él según la situación, pues la mayor parte del tiempo se comportará como una lista, estructura que por lo general no vamos a buscar al querer trabajar con conjuntos de datos ordenados, en formato tabular, mientras que el XML recibido tendrá una estructura jerárquica.

Acudiendo a otra página de ayuda de la BGG XML API [23], nos encontramos con que actualmente, si se envían demasiadas solicitudes el servidor devolverá códigos 500 y 503, indicando que está demasiado ocupado. Para corregir esto, se comenta que una simple espera de 5 segundos entre consultas puede ser suficiente.

Se comenta en la página referenciada, además, que es posible descargar listas de juegos de usuarios en formato CSV, y que se puede acceder a la API mediante Perl o HTML. Sin embargo, no existe ninguna lista de juegos, hecha por un usuario, con todos los juegos de la plataforma, o con un gran número de ellos al menos, luego no nos será

posible recurrir a este método. Especifican, asimismo, que teóricamente también debería admitir el trabajo con PHP.

Por claridad, dejamos aquí un ejemplo de enlace como los que usaremos próximamente, en la siguiente sección:

https://www.boardgamegeek.com/xmlapi/boardgame/1,2?stats=1&comments=1

3.2 Extracción y limpieza de datos

Ahora, dado que sabemos qué información nos reporta la BGG XML API, nos planteamos descargar información concreta sobre los juegos contenidos en su base de datos. Como veremos, este proceso se revela mucho más difícil de lo que parece, al ser la primera ocasión a lo largo del grado en la que hemos trabajado con un conjunto de datos tan grande; con más de 300.000 juegos registrados en la plataforma y más de 25.000 comentarios para ciertos juegos.

En un primer momento, para intentar seguir un orden, buscamos descargar la información de los juegos de la BGG, pues al ser tan extensa y tener la limitación de la longitud máxima de la URL, no nos es posible recuperar con una única búsqueda todos los juegos; y dado que tenemos que respetar los 5 segundos entre búsqueda y búsqueda, parece sensato descargar el contenido de los juegos de mesa, ejecutando cada consulta 5 segundos después de la anterior.

Ahora bien, teniendo en cuenta nuestros intereses, nos centraremos únicamente en guardar las siguientes variables, todas ellas guardadas como el valor (casi siempre) de un nodo hijo del padre *boardgame*, o de un nieto. Hacemos la siguiente separación, para comentar los primeros problemas a los que nos enfrentamos.:

1. Variables almacenadas como nodos en el XML y con un único valor: *yearpublished*, *minplayers*, *maxplayers*, *minplaytime*, *maxplaytime*, *age*, *name primary*="*true*" (hay otros nombres) y *description*. Ejemplo:

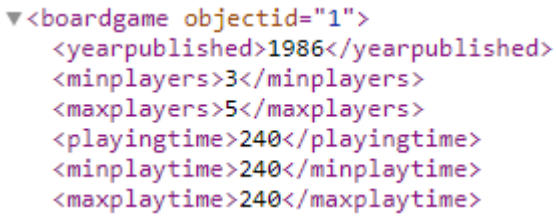

Figura 3.3: Nodo padre *boardgame*.

Fuente: Elaboración propia a partir de consulta a la API de la BGG.

2. Variables que están duplicadas en varios nodos hijos con las mismas características, todos ellos hijos directos del nodo padre *boardgame*: *boardgamemechanic*, *boardgamefamily*, *boardgamecategory* y *comment*. Ejemplos:

```
<boardgamemechanic objectid="2916">Alliances</boardgamemechanic>
<boardgamemechanic objectid="2080">Area Majority / Influence</boardgamemechanic>
<boardgamemechanic objectid="2012">Auction/Bidding</boardgamemechanic>

<comment username="1Aspielerin" rating="6">first edition
<comment username="3greyhounds" rating="7.5">Awesome game
<comment username="4Corners" rating="6">Great theme, very
```

Figura 3.4: Nodos boardgamemechanic y comment.

Fuente: Elaboración propia a partir de consulta a la API de la BGG.

3. Variables que se encuentran diluidas en una cantidad de información mucho mayor de la que podríamos llegar a utilizar, todas ellas relativas a encuestas, y que generalmente están implícitas en nodos con varios hijos: *suggested_numplayers, language_dependence, suggested_playerage*, todas dentro de nodos *result*. Ejemplo:

```
▼<poll name="suggested_numplayers" title="User Suggested Number of Players" totalvotes="128">
  ▼<results numplayers="1">
     <result value="Best" numvotes="0"/>
     <result value="Recommended" numvotes="1"/>
     <result value="Not Recommended" numvotes="80"/>
   </results>
  ▼<results numplayers="2">
     <result value="Best" numvotes="0"/>
     <result value="Recommended" numvotes="1"/>
     <result value="Not Recommended" numvotes="81"/>
   </results>
  ▼<results numplayers="3">
     <result value="Best" numvotes="1"/>
     <result value="Recommended" numvotes="25"/>
     <result value="Not Recommended" numvotes="72"/>
   </results>
  ▼<results numplayers="4">
     <result value="Best" numvotes="23"/>
     <result value="Recommended" numvotes="83"/>
     <result value="Not Recommended" numvotes="9"/>
   </results>
  ▼<results numplayers="5">
     <result value="Best" numvotes="109"/>
     <result value="Recommended" numvotes="12"/>
     <result value="Not Recommended" numvotes="2"/>
   </results>
  ▼<results numplayers="5+">
     <result value="Best" numvotes="1"/>
     <result value="Recommended" numvotes="0"/>
     <result value="Not Recommended" numvotes="58"/>
   </results>
 </poll>
```

Figura 3.5: Nodo padre *poll* con nodos *result* para varias categorías.

Fuente: Elaboración propia a partir de consulta a la API de la BGG.

4. Los nodos hijos del nodo *statistics*: *usersrated*, *average*, *ranks* (otro nodo padre con varios nodos hijo *rank*), *owned*, *trading*, *wanting*, *wishing*, *numcomments*, *numweights*, *averageweight*

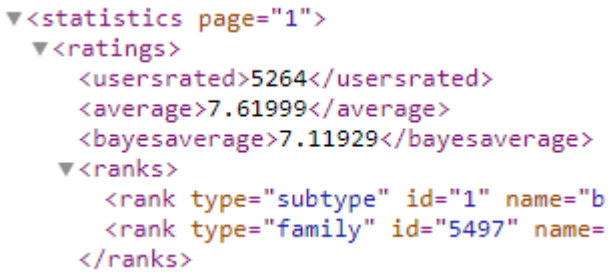

Figura 3.6: Extracto del nodo padre *statistics*.

Fuente: Elaboración propia a partir de consulta a la API de la BGG.

Veamos en detalle un rango, expresado aparte por la longitud del nodo:

```
<rank type="subtype" id="1" name="boardgame" friendlyname="Board Game Rank" value="296" bayesaverage="7.11929"/>
```

Antes de seguir, aprovechamos este momento para comentar diversos problemas de almacenamiento que padece la BGG en cada una de las categorías construidas, y que por ende nos dificultarán la tarea de extraer conjuntos ordenados de datos a apartir de ellos.

1. Refiriéndonos a las variables almacenadas como *yearpublished* o *minplayers*, valores de un nodo, podemos comentar que al encontrarse vacías almacenan la información como entradas de tipo <nombre/>, dificultándonos la tarea de leer los datos. Además, se añade otro problema, que codifican la ausencia de año publicado de dos formas, como hemos dicho y como un nodo *yearpublished* con el valor cero en su interior. Este proceder no es consistente, y al ser algo poco esperable nos dimos cuenta del problema tarde, si bien la corrección era relativamente sencilla, como veremos en la siguiente sección. Por otro lado, refiriéndonos ahora a los nombres, parece importante destacar que cada juego dispone de varios nombres en múltiples idiomas o por cualquier otra cuestión, y cada nodo de tipo nombre se encuentra al mismo nivel como hijo directo de *boardgame*, el nodo padre, pudiéndolos diferenciar gracias al atributo. Este es el primer ejemplo de mala organización que veremos, pues lo más correcto sería tener los diferentes nodos como hijos de un nodo padre de tipo nombres, quizás incluso suprimiendo sus atributos y cambiándolos por el nombre del nuevo nodo, lo que le daría una estructura más ordenada al almacenamiento de esta variable.

2. Considerando ahora las variables que son del mismo tipo de nodo pero que se encuentran todas al mismo nivel, como *boardgamemechanic* o *comment*, esto supone una dificultad añadida en un primer momento, al visualizar los datos, al igual que a la hora de leerlos, pues lo lógico sería que hubiese nodos padres *boardgamemechanics* y *comments*, de los cuales los ya mencionados fuesen hijos. De esta forma la lectura e interpretación resultaría mucho más sencilla. Cabe destacar que los nodos *boardgamemechanic* no se imprimen en pantalla necesariamente juntos, y que lo habitual es encontrarlos entre cientos de nodos sobre episodios de podcast, todos ellos al mismo nivel que *boardgamemechanic*, hijos directos de *boardgame*. Adicionalmente, dentro de *boardgamefamily* nos encontramos con una familia que es "*Admin: Better Description Needed!*", lo cual para el analista puede llegar a resultar confuso, al ser información que uno no esperaría encontrar en una lista de familias para un juego, y es que se trata de una nota para los propios administradores de la plataforma, comentario que debería encontrarse quizás en un nodo aparte, en una sección de notas para administradores o en un documento aparte, pero no en una lista de familias a las que pertenece un juego.
3. Centrándonos ahora en las encuestas, en un primer momento nos podríamos plantear el extraer el resultado de una encuesta dada, obteniendo, por ejemplo, las siguientes variables: *numplayers_best*, *numplayers_recommended* y *numplayers_not_recommend*; y análogamente para las otras dos variables asociadas a encuestas. Sin embargo, mirando el documento nos encontramos con que esta información adicional no se encuentra representada en ninguna parte, y por lo tanto es necesario ejecutar una búsqueda para obtenerla. Adicionalmente, y aquí se encuentra el problema a la hora de almacenar los datos, tanto los resultados para cada posible valor de la encuesta, como el valor en cuestión, se encuentran almacenados como **atributos** dentro de sendos nodos *results* y *result*, en lugar de ser valores de nodos. Esto claramente va en contra de los principios de XML de diferenciación etiqueta-valor, y sería conveniente que el nodo *results* tuviese como valor el número al que hace referencia, y análogamente con los diferentes nodos *results*. De esta forma suprimiríamos los atributos de los nodos.
4. Por último, refiriéndonos ahora a las estadísticas, aunque podamos argumentar que sería mejor guardar *rank* como un nodo con un valor no establecido como atributo, el problema en este punto es diferente, y es que no aparece documentado en parte alguna del manual el significado de los diferentes valores

que devuelve el parámetro *stats* a la hora de realizar una consulta con la API, siendo por lo tanto imposible interpretar el valor de varios nodos, como por ejemplo *bayesaverage*.

Para el lector, queda aquí un ejemplo de juego con entradas vacías:

https://www.boardgamegeek.com/xmlapi/boardgame/133305?comments=1&stats=1

Adicionalmente y en relación con la ayuda de la API, en ningún lugar se comenta cómo se asignan las id de los juegos pero, tal y como comprobamos empíricamente, parece que se asignan de forma automática, sin posibilidad de borrarse o de cambiarse. Esto es relevante pues, tal y como veremos a continuación, es importante saber que del primer juego al último no hay ninguna id vacía.

Antes de abordar el problema de la descarga de los datos de la BGG, hay que abordar otro primero, y es que no existe ningún dato ni en el manual ni en las referencias de la BGG sobre el número de juegos que hay en su base de datos, razón por la cual nos vemos obligados a ejecutar una búsqueda dicotómica (para poder calcular el número total de juegos y poder acotar así la dirección a la haremos las peticiones), que parta de una pareja de ids para la que sepamos que hay una entrada para la primera, y que no hay para la segunda. Dada esta pareja, el algoritmo toma un valor intermedio y comprueba si la entrada está vacía o no, asigna la posición a aquella que comparte este atributo con ella y va iterando hasta que llega a dos valores consecutivos. De esta forma, sabemos que el primer valor es el último juego de la base de datos, y por lo tanto es el número de juegos disponibles en la BGG. El punto principal de la búsqueda es el siguiente, teniendo en cuenta que *lim_bgg* es la pareja mencionada, iniciada en un valor conveniente como la pareja $10^5, 10^6$:

```
while(abs(lim_bgg[1] - lim_bgg[2]) > 1) {
  n_lim_bgg = round(mean(lim_bgg))
  n_bgg_game_url = get_game(n_lim_bgg)
  n_is_empty_bgg = n_bgg_game_url %>%
    read_html(as.data.frame = T, stringsAsFactors = TRUE) %>%
    html_nodes("error") %>% length == 1
  if (n_is_empty_bgg) {
    lim_bgg[2] = n_lim_bgg
    cat("Cambiamos la cota superior por:", n_lim_bgg, "\n")
  } else {
    lim_bgg[1] = n_lim_bgg
    cat("Cambiamos la cota inferior por:", n_lim_bgg, "\n")
  }
}
```

Figura 3.7: Punto principal a la hora de averiguar la longitud de la base de datos.

Fuente: Elaboración propia.

Notemos que hemos definido *get_game* como una función que proporciona la url de un juego en la API dada su id, y que la línea que se inicia *n_is_empty_bgg* lee simplemente

si hay un error o no, devolviendo un nodo vacío o un nodo y comparando la longitud del resultado con 1, obteniendo de esta forma una verificación de si está vacío o no.

Al ejecutar la función a principios de marzo comprobamos que el algoritmo devolvía la pareja de valores 332.030 y 332.031. Ahora bien, como la BGG se actualiza prácticamente de forma diaria, fijamos este valor para el resto del trabajo, ignorando nuevos juegos incluidos en la base de datos durante los últimos dos meses.

Conocido ya el tamaño de la BGG con el que vamos a trabajar, el siguiente paso es descargar toda la información relativa a los juegos con id entre 1 y 332.030. Para ello, como las url tienen un número máximo de caracteres y la BGG XML API nos obliga a esperar 5 segundos entre consulta y consulta, planteamos el descargar toda la base de datos en 1000 iteraciones de un bucle, descargando de esta forma unos 300 juegos en cada paso, que guardaría en un documento XML en una carpeta prefijada, *data/bgg*. Incluimos lo principal de esta idea, ejecutada para cada iteración k, teniendo en cuenta que se ejecuta *Sys.sleep(5)* de una iteración a otra:

```
n0 = jump_size * k
n1 = n0 + jump_size
bgg_url_games = paste0(paste0(c(paste0(bgg_url_base, n0), (n0+1):n1), collapse = ","),
                       "?comments=1&stats=1")
root <- xml_new_document() %>% xml_add_child(.value = "games")

cat("\t * Lanzamos la consulta ", k, "-ésima...\t", sep = "")
bgg_games_bg <- bgg_url_games %>% read_xml %>% xml_find_all("boardgame")
cat("Consulta ", k, "-ésima realizada.\n", sep = "")
l_bgg_games_bg = length(bgg_games_bg)

cat("\t * Preparamos la escritura del documento ", k, "-esimo...\t", sep = "")
  for (i in seq(l_bgg_games_bg)) {
    root %>% xml_add_child(.value = bgg_games_bg[[i]]) %>% invisible()
  }
write_xml(root, file = paste0("data/bgg/games_jump_", k, ".xml", collapse = ""),
          options = "as_xml")
cat("Documento escrito: games_jump_", k, ".xml.\nEsperando siguiente iteración...
    \n-------------------------------------------------\n", sep = "")
```

Figura 3.8: Fragmento de *bgg_jump*, la función de escritura.

Fuente: Elaboración propia.

Notemos, primeramente, que n0 y n1 son las ids que recorrerá nuestra función en cada iteración. Por otro lado, desgraciadamente, debemos ejecutar cada iteración dentro de un condicional *tryCatch*, pues determinados valores de las ids proporcionar un error que no hemos logrado identificar a pesar de ejecutar la orden de forma separada. Dado que solo nos devuelve el error para 10 valores de k, no les damos más importancia, aunque perdamos de esa forma, teóricamente al menos, una de cada 30 entradas de la BGG. Adicionalmente, en la sección de error le pedimos a la función que guarde los valores

de los k que nos han dado errores, para tenerlos en cuenta a la hora de leer los documentos.

De esta forma hemos logrado descargar los archivos en formato XML, siendo todos los juegos, dentro de cada archivo *game_n.xml*, un hijo del nodo padre *games*. Resulta relevante remarcar que la XML BGG API no devuelve los resultados en el acto, lo que sumado al tiempo de ejecución natural de la función da lugar a un tiempo de ejecución de casi medio minuto entre iteración e iteración.

El resultado logrado, en definitiva, es una carpeta en la que tenemos almacenados casi 1000 documentos XML en los que tenemos almacenada la información de la mayor parte de los juegos de la BGG. Adicionalmente, hemos guardado los índices de los documentos para los cuales no disponemos de documentos en formato XML asociados. Es interesante destacar que los juegos finalmente ocupan 1.33 GB, un tamaño más que considerable con el que trabajar en los siguientes pasos.

3.2.1 Preparación y selección de datos de la BGG

Una vez hemos descargado los datos, necesitamos ahora cargarlos a nuestro entorno de trabajo, lo cual incluye un gran número de problemas añadidos debido a la forma en la que se almacenan los datos en la BGG, como ya mencionamos en la sección anterior. Si bien podríamos haber limpiado los datos al tiempo que los descargábamos, vimos más conveniente dividir el trabajo, descargando primero los datos, leyéndolos y limpiándolos después y guardándolos como archivos RData en último lugar, para poder trabajar de forma directa con ellos en varias ocasiones.

En esta sección, por lo tanto, nos centramos en la lectura de los datos, intentando darles formato al mismo tiempo que los leemos de los diferentes archivos en formato XML que hemos generado en el punto anterior. Por lo tanto, nos centraremos en los puntos clave de la función que hemos creado, *read_bgg*, que combina herramientas de `dplyr`, `xml2` y `magrittr`, haciendo un uso extensivo de todas ellas.

Notemos que, como descargamos directamente los documentos XML sin limpiarlos, a la hora de leerlos tendremos que emplear tiempo adicional, así como un mayor número de líneas de código.

Primeramente destacamos que, como ocurrió antes con la función de escritura, nos regiremos por la buena praxis de mostrar en pantalla los resultados que vamos obteniendo, siendo finalmente el resultado obtenido algo semejante a lo siguiente durante la ejecución del código de lectura:

```
        Algoritmo iniciado a las 23:12 con 999 elementos que leer.
        Con 50 segundos de ejecución por lectura, estimamos que el proceso
 durará: 832.5 minutos.
-----------------------------------------------------
ITERACIÓN 1
        * Lanzamos la lectura 1-ésima...       Lectura 1-ésima realizada
 con 303 juegos.
        * Campos leídos: yearpublished, minplayers, maxplayers, minplayti
me, maxplaytime, age, name, description, boardgamemechanic, boardgamefamil
y, boardgamecategory, numplayers, language_dependence, suggested_playerag
e, comment, usersrated, average, ranks, owned, trading, wanting, wishing,
 numcomments, numweights, averageweight.
Lectura y procesado correctos. Esperando siguiente iteración...
-----------------------------------------------
ITERACIÓN 2
```

Figura 3.9: Salida por consola de *read_bgg*, indicando iteraciones y campos leídos.

Fuente: Elaboración propia.

En este caso, dado que el tiempo de lectura era prácticamente constante, optamos por estimar de forma empírica el valor de ejecución de cada lectura, asignándole el valor estimado de 50 segundos. Por otro lado, que los nombres de los campos se impriman por pantalla al tiempo que se ejecuta la función nos ha permitido suprimir algunas variables en varias ocasiones para obtener diferentes conjuntos de datos de generación y manipulación más rápidos, al poder ver, empíricamente, a qué variables le dedicaba más tiempo la lectura, siendo las más pesadas las diferentes categorías y los comentarios, para tamaños superiores al centenar, pues pusimos el número de comentarios como una variable, como comentaremos próximamente.

Ahora bien, el procedimiento de lectura de los diferentes campos varía considerablemente de un campo a otro, y es por ello que destacamos los diferentes procedimientos realizados sobre los datos descargados:

```
yearpublished <- bgg_games_raw %>%
  xml_find_all("//yearpublished") %>%
  xml_text() %>%
  as.integer()
yearpublished[yearpublished == 0] = NA
```

Figura 3.10: Extracción y corrección del campo *yearpublished* en la función *read_bgg*.

Fuente: Elaboración propia.

Aquí observamos el primer acceso a una de las variables mejor guardadas que tenemos dentro del archivo leído, que se ha cargado al entorno como *bgg_games_raw*. Es importante remarcar que, como destacamos antes, el año de publicación se había

codificado de dos formas diferentes en la BGG: como 0 y como NA, y por lo tanto decidimos recodificarlo, unificándolo todo como NA.

```
name <- bgg_games_raw %>% xml_find_all("//name")
name %<>% extract(!is.na(xml_attr(name, "primary") == "true"))
name %<>% xml_contents() %>% xml_text()
```

Figura 3.11: Extracción de los nombres primarios de los juegos en la función *read_bgg*.

Fuente: Elaboración propia.

A la hora de escoger el nombre en los resultados disponemos de varios nodos name con atributos, todos al mismo nivel. Recordemos de forma breve que era necesario diferenciar los nombres al estar todos al mismo nivel, pudiéndose caracterizar gracias al atributos *name*, que notemos que es también el nombre del nodo, una mala decisión que podría dar lugar a problemas de programación o simplemente de comprensión.

Nótese el uso de la tubería especial "*%<>%*" de `magrittr`, que además de pasar el dato actualiza el conjunto al que se aplica.

```
boardgamemechanic <- list()
for (i in seq(n_bgg_games)) {
  boardgamemechanic[[i]] <- bgg_games_raw %>%
    xml_children() %>%
    extract(i) %>%
    xml_find_all(".//boardgamemechanic") %>%
    xml_contents() %>%
    as_list()
}
```

Figura 3.12: Extracción de las diferentes mecánicas de cada juego en *read_bgg*.

Fuente: Elaboración propia.

Acudiendo ahora a las mecánicas y pudiendo extender esto para las categorías y las familias, estas se encuentran repartidas a lo largo de varios nodos al mismo nivel, razón por la cual nos vemos obligados a, dado el juego i-ésimo extraído de *bgg_games_raw*, encontrar todos los nodos del tipo seleccionado del nodo padre, convirtiéndolos de forma inmediata en lista. Aunque no obtengamos de forma directa un conjunto ordenado de datos, el guardarlo como lista y no duplicar o triplicar las filas asociadas al juego facilitarán la lectura al ocupar de esta forma menos memoria el objeto generado.

```
xml_polls = bgg_games_raw %>% xml_find_all("//poll")
xml_polls_sn = xml_polls[xml_polls %>%
                           xml_attr("name") == "suggested_numplayers"]
for (i in seq(n_bgg_games)) {
  nodeset_i = xml_polls_sn[i] %>% xml_children()
  l_nset_i = length(nodeset_i)
  exit = data.frame(matrix(nrow = 3, ncol = l_nset_i),
                    row.names = c("Best", "Recommended", "Not Recommended"))
  colnames(exit) = nodeset_i %>% xml_attr("numplayers")
  for (j in seq(l_nset_i)) {
    exit[,j] = nodeset_i[j] %>% xml_contents() %>% xml_attr("numvotes")
  }
maxs = exit %>% apply(FUN = max, MARGIN = 1)
numplayers_best %<>% c(which(exit[1, ] == maxs[1])[1])
```

Figura 3.13: Extracción de las variables derivadas de los diferentes nodos *poll*.

Fuente: Elaboración propia.

Centrándonos ahora en campos cuyos valores no tenemos en un primer momento, sino que tenemos que generar recorriendo un nodo padre, a la hora de trabajar con los resultados de las diferentes encuestas debemos tener un especial cuidado, pues la API nos devolvió no solo, por ejemplo, el número recomendado por los usuarios de la BGG de jugadores recomendados, sino que también nos proporcionó el número de votos de cada posible votación relacionada. Algo más en detalle, esto significa que, siguiendo con el ejemplo propuesto, dado que el número recomendado de jugadores varía entre 2 y 8 por lo general, tenemos varios nodos con el número de votos para cada uno de esos valores dentro de nodos reservados para *best*, *recommended* y *not recommended*. Aunque no suponga una dificultad el obtener esta información al poder hacerse de muchas formas, dado que en cada salto de *read_bgg* leemos unos 300 juegos, hay que notar que esta versión depurada de la función sigue tardando bastante en este punto, pues para cada juego crea una matriz de 3x8 aproximadamente, generando de esta forma 7.200 entradas de matriz en cada iteración de *read_bgg*. Una forma de mejorar el rendimiento sería suprimir la matriz, quedándonos únicamente con el máximo de cada categoría en un paso.

```
for (i in seq(n_bgg_games)) {
  node <- bgg_games_raw %>% xml_children() %>%
    extract(i) %>% xml_find_all("//comment")

  node %<>% sample(node, size = min(length(nodo), comment_sample_size))
  comment[[i]][["rating"]] <- nodo %>% xml_attr("rating")
  comment[[i]][["text"]] <- nodo %>% xml_contents() %>% xml_text()
}
```

Figura 3.14: Extracción de los comentarios para un juego dado en *read_bgg*.

Fuente: Elaboración propia.

Acudiendo ahora a los comentarios, y extendiendo los comentarios ya realizados sobre la forma de almacenar la información de la BGG, en este caos vemos que el valor de la puntuación del comentario se encuentra almacenado como atributo, lo que nuevamente es una forma poco adecuada de almacenar esta información, tal y como se vio antes. Dado que usaremos los comentarios de forma muy separada del resto, no nos resulta un problema guardarlos como una lista, como antes hicimos con las categorías, pues los ordenaremos a la hora de trabajar con ellos.

De esta forma hemos logrado crear una función que, dados un número de documentos y un tamaño muestral para los comentarios nos proporciona un *tibble* con todas las variables comentadas, siendo un ejemplo de salida:

name	boardgamecategory	yearpublished	comment	language_dependence
Die Macher	*list("Economic", "Neg...*	1986	*list(1, text = c("Only ...*	No necessary in-game text
Dragonmaster	*list("Card Game", "Fa...*	1981	*list(2, text = c("Some...*	Some necessary text
Samurai	*list("Abstract Strategy...*	1998	*list(3, text = c("DIY v...*	No necessary in-game text
Tal der Könige	*list("Ancient")*	1992	*list(4, text = c("2-6 p...*	No necessary in-game text
Acquire	*list("Economic", "Terri...*	1964	*list(5, text = c("Abstr...*	No necessary in-game text
Mare Mediterraneum	*list("Civilization", "Na...*	1989	*list(6, text = c("Playe...*	Some necessary text
Cathedral	*list("Abstract Strategy")*	1978	*list(7, text = c("Wolf...*	No necessary in-game text
Lords of Creation	*list("Civilization", "Fa...*	1993	*list(8, text = c("Very ...*	No necessary in-game text
El Caballero	*list("Exploration")*	1998	*list(9, text = c("nice l...*	No necessary in-game text
Elfenland	*list("Fantasy", "Travel")*	1998	*list(10, text = c("This...*	No necessary in-game text
Bohnanza	*list("Card Game", "Fa...*	1997	*list(11, text = c("An i...*	No necessary in-game text
Ra	*list("Ancient", "Mythol...*	1999	*list(12, text = c("5", "...*	No necessary in-game text
Catan	*list("Economic", "Neg...*	1995	*list(13, text = c("Oh ...*	Some necessary text

Figura 3.15: Ejemplo de salida de la función *read_bgg*.

Fuente: Elaboración propia.

4 ANÁLISIS EXPLORATORIO Y VISUALIZACIÓN

Nuestro objetivo en este capítulo es estudiar los datos que hemos generado, ya sea mediante técnicas estadísticas puramente descriptivas o mediante el análisis gráfico, resumiendo en ambos casos la información contenida en los objetos, para las variables más relevantes, con las que trabajaremos en los siguientes capítulos. Además de conocer de esta forma la estructura de los conjuntos, buscamos detectar comportamientos y patrones que nos ayuden a orientar el proyecto, como por ejemplo saber qué variables influyen más o de qué forma, o si existe algún comportamiento generalizado a lo largo del tiempo o respecto de cierta variable prefijada.

4.1 Análisis descriptivo

Primeramente hemos de decir que con la función de lectura anterior, *read_bgg*, logramos generar hasta 300.404 entradas para el objeto cargado, esto es, disponemos de la información indicada, un total de 25 variables, para cada uno de esos juegos, si bien en algunas ocasiones nos encontraremos con valores NA como consecuencia de la ausencia de los mismos en la base de datos. Por otro lado, otro punto que hay que remarcar es que todas las entradas tienen siempre nombres diferentes, aunque puedan ser expansiones unos de otros, proporcionándonos de esta forma un identificador para cada posible juego en el objeto creado.

```
Rows: 2,720
Columns: 25
$ name                      <chr> "Die Macher", "Dragonmaster", "Samurai"
$ yearpublished             <int> 1986, 1981, 1998, 1992, 1964, 1989, 197
$ minplayers                <int> 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3,
$ maxplayers                <int> 5, 4, 4, 4, 6, 6, 2, 5, 4, 6, 7, 5, 4,
$ minplaytime               <int> 240, 30, 30, 60, 90, 240, 20, 120, 90,
$ maxplaytime               <int> 240, 30, 60, 60, 90, 240, 20, 120, 90,
$ boardgamemechanic         <list> ["Alliances", "Area Majority / Influen
$ boardgamefamily           <list> ["Country: Germany", "Political: Elect
$ boardgamecategory         <list> ["Economic", "Negotiation", "Political
$ numplayers_best           <int> 4, 4, 4, 3, 5, 4, 2, 4, 2, 4, 7, 5, 5,
$ numplayers_recommended    <int> 4, 3, 3, 4, 4, 6, 2, 3, 2, 6, 2, 5, 3,
$ numplayers_not_recommended <int> 4, 1, 4, 1, 6, 2, 1, 1, 2, 1, 6, 3, 2,
$ language_dependence       <chr> "No necessary in-game text", "Some nece
$ suggested_playerage       <chr> "14", "2", "10", "12", "12", "12", "8",
$ comment                   <list> [1, <"Only played once. The ability to
$ usersrated                <int> 5250, 557, 14871, 338, 18300, 81, 3251,
$ average                   <dbl> 7.62256, 6.64398, 7.45017, 6.59769, 7.3
$ ranks                     <list> [<"Board Game Rank", "Strategy Game Ra
$ owned                     <int> 7295, 1263, 15325, 636, 23255, 125, 585
$ trading                   <int> 234, 62, 295, 40, 878, 2, 267, 30, 182,
$ wanting                   <int> 500, 72, 782, 55, 531, 32, 99, 21, 87,
$ wishing                   <int> 2010, 190, 3376, 122, 2611, 49, 569, 55
$ numcomments               <int> 1993, 307, 3760, 120, 5770, 27, 1217, 8
$ numweights                <int> 758, 54, 1442, 30, 1600, 7, 277, 15, 14
$ averageweight             <dbl> 4.3245, 1.9630, 2.4882, 2.6667, 2.5038,
```

Figura 4.1: Resultado de aplicarle *glimpse* al objeto creado, para los primeros 2720 juegos.

Fuente: Elaboración propia.

De estas 25 variables, una es el nombre, la id, como decíamos, 2 son variables categóricas, 5 son listas que hay que expandir para poder trabajar con ellas y 17 son numéricas, lo que nos proporciona un gran punto de partida a la hora de continuar con el proyecto, pues tenemos desde contenido al que aplicarle las herramientas de procesamiento del lenguaje natural hasta la posibilidad de construir multitud de gráficos para analizar diversos aspectos sobre las variables cuantitativas. Teniendo esta idea en mente, desarrollaremos en esta sección el análisis descriptivo de los datos generados, lo que nos servirá para entender mejor la forma de los datos con que contamos y para plantear estudios y modelos posteriores aprovechando la información disponible.

Como ya mencionábamos, las dimensiones de los datos obtenidos nos limitan a la hora de llevar a cabo estudios más en profundidad, obligándonos a trabajar en la mayor parte de los casos con subconjuntos de los que extraer resultados. Sin embargo, como veremos a lo largo de este capítulo, *bgg_*games, el resultado de leer los primeros 10 documentos XML generados con *read_bgg*, contiene 2720 entradas que, como comprobaremos, representan bastante bien el comportamiento general de los datos. Dicho esto, estudiemos sobre el conjunto mencionado, primeramente, las variables cuantitativas con las que trabajaremos:

	min	median	max	mean	var
yearpublished	-3500	1992	2012	1976.250646	44634.69
minplayers	0	2	8	2.134367	0.39
maxplayers	0	5	100	5.479144	50.26
minplaytime	0	45	2480	76.720930	8679.07
maxplaytime	0	60	12000	89.693614	77766.12
numplayers_best	1	2	18	2.688815	2.69
numplayers_recommended	1	2	17	2.221853	2.27
numplayers_not_recommended	1	1	21	1.684016	2.59
usersrated	0	125	103682	982.172019	18851681.95
average	0	6	9	5.817042	1.22
owned	0	317	157551	1599.384275	40716800.09
trading	0	18	2466	53.050203	14713.14
wanting	0	7	1369	31.676264	8048.89
wishing	0	23	10316	122.055371	240970.94
numcomments	0	60	18719	287.918051	845070.98
numweights	0	13	7545	81.463640	107935.74
averageweight	0	2	5	1.839662	0.84

Figura 4.2: Análisis descriptivo para las variables de los primeros 2720 juegos.

Fuente: Elaboración propia.

Vemos que disponemos de dos conjuntos de variables, agrupadas por órdenes, pues es inmediato comprobar que tenemos campos que presentan un rango elevado, como *yearpublished* – que tiene como mínimo -3.500 – o *usersrated*, pero también disponemos de categorías en las que, por aquello que medimos, no es posible una gran

variación, como ocurre con *minplayers* o *averageweight*, donde los valores se encuentran siempre entre 0 y 10. En general, esta variación justificará el uso de herramientas de normalización dentro de las recetas de las diferentes secciones del capítulo 6, evitando de esta forma que los órdenes de magnitud influyan en los resultados.

Recordamos en este punto que la ausencia de valores se ha almacenado en muchos casos como valores nulos, heredando esto de la BGG, si bien esto no siempre representa que realmente el valor esté perdido, como por ejemplo en *averageweight*, donde los pesos se asignan entre 1 y 5 o en average [24], o en *minplayers* y *minplaytime*, donde la casuística es semejante. Sin embargo, en *usersrated*, *numcomments* o *wanting* realmente el valor nulo puede tener sentido.

En general, nos encontramos con que para variables de recuento, como *numcomments* o *numweights*, la mediana es muy pequeña, mientras que el máximo es desmedido, lo que nos indica que en la sección de la minería de texto, donde trabajaremos con un subconjunto de estos datos, los resultados que obtengamos podrían no ser demasiado exactos al ser el número de valoraciones muy diferente entre juegos. Sin embargo, dado que presentaremos resultados para un subconjunto de juegos, dado que nuestra selección contendrá suficientes comentarios de cada, no supondrá un problema.

De cualquier forma, disponemos de un conjunto amplio de variables cuantitativas y, aunque deberíamos sustituir los valores de las variables problemáticas mencionadas por NA cuando tienen el valor 0, lo dejaremos pendiente para cuando trabajemos con los modelos en el capítulo 6.

Centrándonos ahora en algunas de las variables cualitativas, empecemos con un recuento de categorías dentro de los campos *suggested_playerage* y *language_dependence:*

```
$suggested_playerage

  2   3   4   5   6   8  10  12  14  16  18 21+
343  13  28  63 166 462 384 338 116 132   5   3

$language_dependence

     No necessary in-game text           Some necessary text
                          1058                           305
        Moderate in-game text          Extensive use of text
                           343                           207
Unplayable in another language
                            68
```

Figura 4.3: Frecuencias de las variables *suggested_playerage* y *language_dependence*, para los 2720 primeros juegos.

Fuente: Elaboración propia.

Estas dos variables fueron las que construimos, extrayéndolas de las diferentes votaciones en las que se encontraban, y vemos que hemos obtenido resultados muy diferentes para cada una, siendo lo más interesante que un gran número de juegos de la base de datos se encuentran enfocados a un público mayoritariamente adolescente, y que la inmensa mayoría de los juegos tienen poco texto o no necesitan directamente texto. Esto será analizado, para un conjunto mayor de datos, en la siguiente sección, donde haremos representaremos gráficamente los resultados de esta tabla, de forma tal que queden más claros, expresados en relación con los otros valores de la categoría mencionada.

Incluimos ahora una tabla para las categorías de juegos más comunes, esto es, los valores contenidos en las listas almacenadas en el campo *boardgamecategory*. Dado que disponemos de más de 100 categorías diferentes y es imposible representarlas todas sin un procesamiento previo, seleccionamos las 10 más comunes, restringiéndonos a ellas. De esta forma, obtenemos la siguiente tabla:

boardgamecategory <chr>	n <int>
Card Game	638
Wargame	491
Abstract Strategy	331
Economic	283
Fantasy	262
Science Fiction	231

Figura 4.4: Categorías más frecuentes en *boardgamecategory* en los primeros 2720 juegos.

Fuente: Elaboración propia.

Observamos que existe una gran variación, incluso dentro de los primeros juegos, pues, por ejemplo, la categoría más común, *Card Game*, aparece más del doble de veces que la cuarta, *Economic.* Esto nos hace pensar en que existe un comportamiento generalizado dentro del mercado, que intenta darle más peso a determinadas categorías, las que podrían ser las más queridas dentro del público, lo cual podría quedar representado en la obtención de diferentes puntuaciones dependiendo de la categoría. Este punto será analizado en detalle en la sección 6.3, donde veremos no solo la importancia de cada categoría en frecuencias absolutas, sino también el efecto de cada una en las valoraciones obtenidas.

Por último, en relación con esta variable, una de las más tratadas a lo largo del trabajo, destacamos que cada juego tiene asociada, como *boardgamecategory*, una lista, pudiendo pertenecer de esta forma a uno o más juegos. Esto dificultará la tarea de construir modelos de predicción, pues habremos de modificar considerablemente los datos hasta obtener un conjunto adecuado para nuestro estudio, como haremos en la sección 6.1, no pudiendo aplicar, por ejemplo, la función *step_dummy* del paquete `recipes` para obtener una matriz de diseño más apropiada.

Siguiendo la idea expuesta de la última figura, intentamos ver cuántas categorías hay para las tres variables *boardgamecategory*, *boardgamefamily* y *boardgamemechanic*, de forma tal que nos hagamos una idea de cómo habremos de tratarlas más adelante:

Variable <chr>	n <dbl>
boardgamecategory_all	84
boardgamefamily_all	4270
boardgamemechanic_all	182
boardgamecategory	82
boardgamefamily	1207
boardgamemechanic	160

Figura 4.5: Número de categorías en *boardgamefamily, boardgamemechanic* y *boardgamecategory,* para los primeros 2720 juegos y para todos los juegos.

Fuente: Elaboración propia.

Representamos en esta figura el número de niveles para cada uno de los factores mencionados, para la muestra de 2720 juegos y para el total. Tal y como vemos, disponemos de un elevado número de categorías para cada variable, siendo especialmente grande para *boardgamefamily*, lo que dificultará la creación de variables *dummy* más adelante, obligándonos a hacer una selección de familias más comunes o

importantes, como haremos en la sección 6.1. Lo mismo se puede aplicar a las otras dos variables, y es que esto nos obligará a trabajar con un subconjunto de los niveles para cada variable, agrupando los que no sean seleccionados en un nuevo nivel “otros”, para favorecer la interpretación de los resultados obtenidos, tanto en los gráficos de la siguiente sección como en los modelos del capítulo 6.

Por otro lado, y siguiendo lo que ya comentamos para la variable *boardgamecategory*, recordemos que cada juego puede tener no solo varias categorías, sino también varias familias y mecánicas, hecho que en los siguientes capítulos dificultará no solo la construcción, sino también la interpretación de los modelos.

Por últimos, vemos que salvo para las familias, en la muestra de 2720 juegos está representada la mayoría de los niveles de las variables *boardgamecategory* y *boardgamemechanic*, lo que nos proporciona cierta confianza a la hora de trabajar con la muestra seleccionada, al menos en lo referido al problema de la representatividad por categorías y mecánicas.

4.2 Análisis gráfico de datos

Nos centramos ahora en algunas de las variables que más nos interesan, como el promedio de la puntuación o su evolución a lo largo del tiempo, así como algunos gráficos para *language_dependence* y series temporales, para obtener ideas que podamos aprovechar más adelante en el capítulo 6, de modelización.

Sin embargo, es importante destacar que en esta sección no se abordará la representación de los comentarios de los juegos, dejando ese punto para el capítulo 5, de procesamiento del lenguaje natural, al ser algo muy diferente de lo que expondremos en esta sección.

4.2.1 Gráficos de frecuencias

Empezamos analizando la frecuencia de las puntuaciones recibidas por los juegos, *average*, la que es la variable más importante dentro del trabajo, representando el valor de un juego. Para ello, construimos el siguiente histograma con la muestra mencionada:

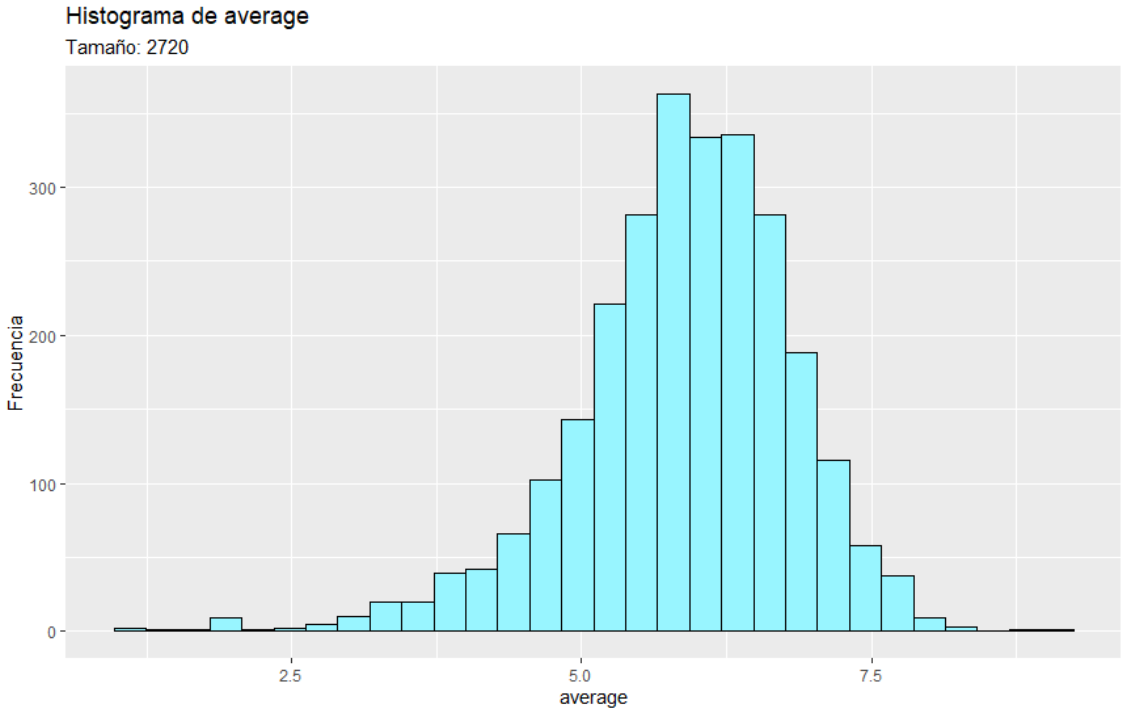

Figura 4.6: Histograma de *average* para los 2720 primeros juegos.

Fuente: Elaboración propia.

Recordemos que, tal y como vimos en la sección 3.1.2, las puntuaciones de los usuarios toman valores enteros entre 1 y 10, convirtiéndose en valores racionales entre 1 y 10 al promediarlos. Vemos, primeramente, que el gráfico obtenido se corresponde con dicha idea, no habiendo ninguna observación por debajo de 1 ni ninguna por encima de 10, y es más, presenta con claridad un comportamiento normal, con una media de 6, correspondiéndose esta con la que ya parecía presentarse en la Figura 4.2. Siguiendo esta idea, parece que la mayor parte de las valoraciones se ubican entre el 5 y el 7.5, siendo raro encontrar una por debajo de 4. Aunque este histograma pueda representar a la población, como es el asociado a las puntuaciones de los primero 2720 juegos almacenados en la plataforma, ocurre algo llamativo, visible en la siguiente figura, en la que se hace la misma representación pero para todos los juegos leídos:

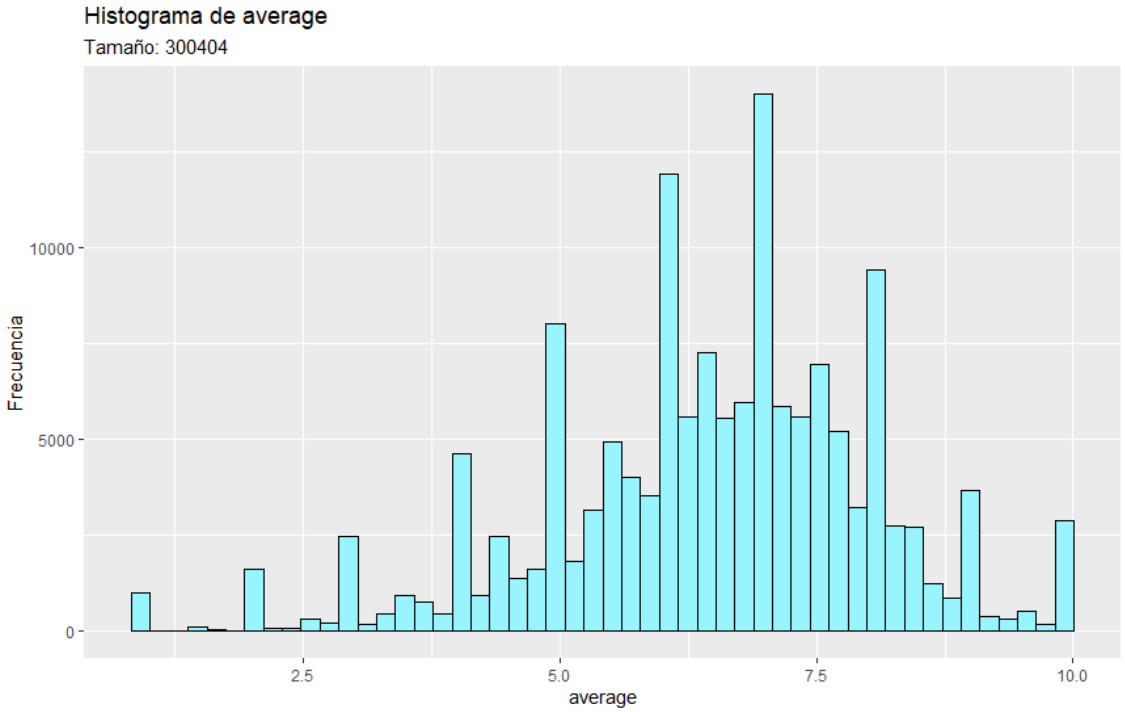

Figura 4.7: Histograma de *average* para todos los juegos.

Fuente: Elaboración propia.

En este caso, aunque que vea un comportamiento semejante al anterior, nos llaman la atención las barras que aparecen equiespaciadas, muy diferenciadas de los valores próximos en cuanto a frecuencia absoluta, y que aparecen incluso en los valores 1 y 10 de *average*. Esta peculiaridad, no observable en el gráfico anterior, tiene una justificación aparentemente sencilla, y es que aquí aparecen juegos añadidos más recientemente a la base de datos de la BGG, así como juegos poco conocidos, que habrán recibido pocas valoraciones. Por lo tanto, juegos con solo una o dos valoraciones tienen con casi toda seguridad un valor entero, correspondiéndose esos valores con los enteros del 1 al 10, lo que ofrece una explicación sobre dichas barras, tal y como podemos ver en la Figura 4.8, donde representamos la distribución de *average* dependiendo del número de votaciones recibidas. Esto nos hace pensar en que, a la hora de trabajar con los datos más adelante, deberíamos restringirnos a juegos con cierto número mínimo de valoraciones, para evitar resultados aberrantes como los que acabamos de observar.

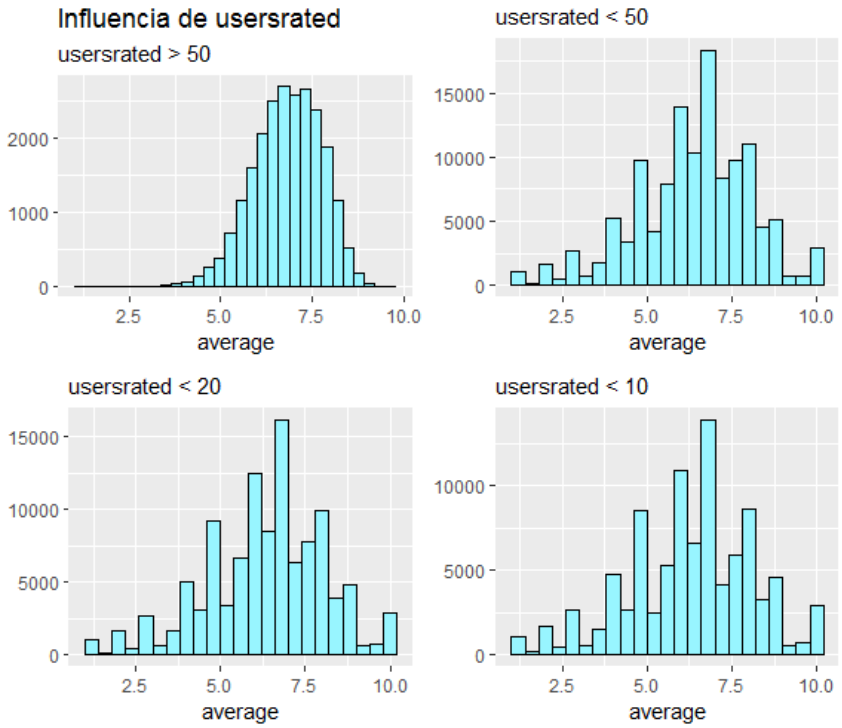

Figura 4.8: Histograma de *average* para todos los juegos, previo filtro con *usersrated*.

Fuente: Elaboración propia.

Otro punto que podemos estudiar, y que además puede proporcionar mucha orientación en lo referido a la posible relación entre las variables cuantitativas, es la matriz de correlación, ofreciéndonos de estar forma información sobre el comportamiento que podrían tener nuestros modelos más adelante al explicar o al menos dar una idea sobre la relación que podría haber entre diversas variables. Por lo tanto, procedemos dando lugar a la Figura 4.9, en la que podemos observar cómo hay un pequeño conjunto de variables con una fuerte correlación, así como otras que parecen presentar cierta relación, si bien de menor valor, como *minplayers* y *maxplayers.*

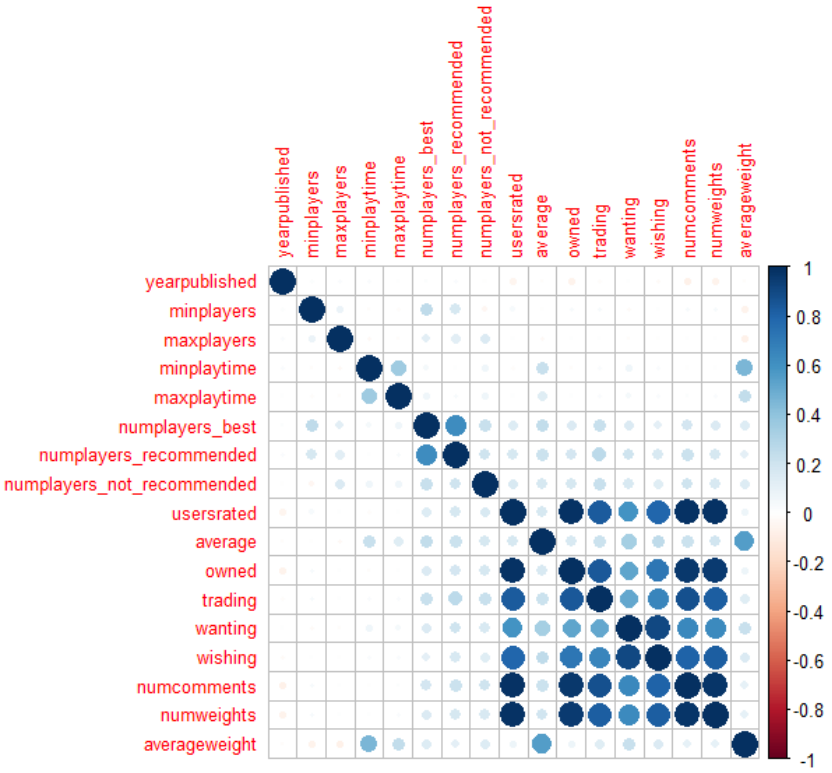

Figura 4.9: Representación de la matriz de correlación con *corplot*, para los primeros 2720 juegos.

Fuente: Elaboración propia.

Nos llaman la atención todas las variables del cuadrante inferior derecho, así como *numplayers_best* y *numplayers_recommended*, pues concretamente estos dos valores deberían proporcionarnos una información semejante, y es por lo tanto esperable que tengan una correlación elevada. Consideramos, por lo tanto, un subconjunto de variables, aquellas que parecen tener una correlación mayor, y representamos nuevamente su matriz de correlación, obteniendo la Figura 4.10.

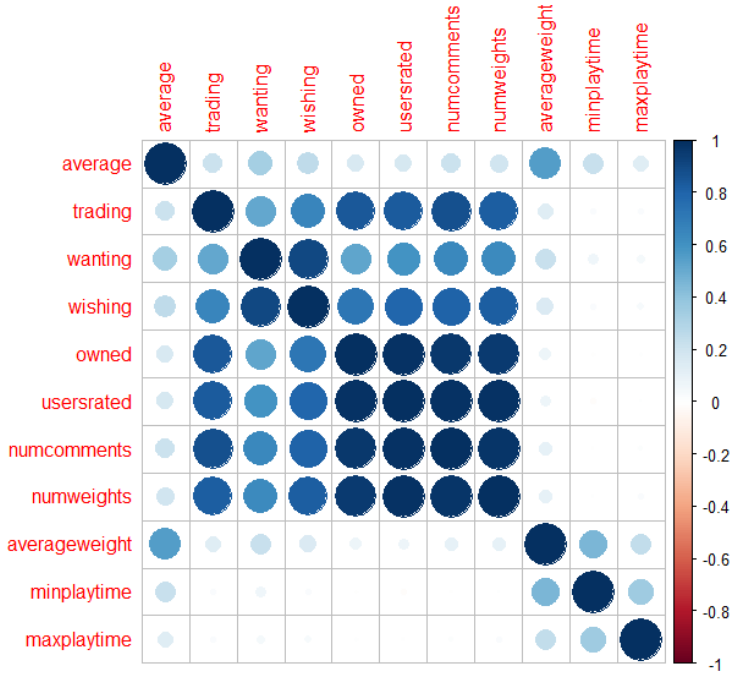

Figura 4.10: Representación de la matriz de correlación con *corplot*, para los primeros 2720 juegos, para variables seleccionadas.

Fuente: Elaboración propia.

A la vista de esta gráfica, es claro que existe una gran correlación entre el número de evaluaciones, de valoraciones de la complejidad, de personas con el juego y de comentarios. Nos llama la atención ver que no existe una correlación negativa entre *trading* y *wishing*, por ejemplo, pues podríamos esperar que si el propietario de un juego quiere intercambiarlo esto podría deberse a que no es demasiado "bueno", y sin embargo esto contrasta mucho con el resultado obtenido, y es que, por lo general, hay más gente queriendo el juego cuanta más gente haya ofreciéndolo, lo que resulta poco esperable.

Refiriéndonos ahora al tiempo y a la dificultad, vemos que, tal y como podíamos esperar, existe correlación positiva entre la duración del juego y la dificultad del mismo, si bien no tan fuerte como para las otras variables mencionadas. Concretamente, para *averageweight* vemos que *average* resulta ser la variable con la que tiene una mayor correlación, dándonos una idea de lo que podríamos esperar, por ejemplo, cuando apliquemos el modelo lineal en la sección 6.2.3.

Adicionalmente, estos resultados nos hacen pensar que el escalamiento multidimensional podría proporcionar grandes resultados, aprovechando la estrecha correlación existente entre un gran número de variables, pudiendo quizás crear, de esta

forma, una variable que represente la “popularidad” del juego, basada en *wanting, wishing, trading* y *owned*, variables que, como hemos visto, presentan una correlación elevada.

```
n_language_dependence_tibble %>%
  ggplot(aes(x = "", y = n, fill = language_dependence)) +
  geom_bar(width = 1, stat = "identity") +
  theme(axis.line = element_blank()) +
  labs(title = "Frecuencias: language_dependence",
       fill = "language_dependence",
       x = NULL, y = NULL) +
  coord_polar(theta = "y", start = 0)
```

Figura 4.11: Código para el diseño de la Figura 4.12, ejemplo de la sintaxis de `ggplot2`.

Fuente: Elaboración propia.

Pasamos ahora a la representación de la variable categórica *language_dependence*, que tiene interés por sí misma, pues nos proporciona información muy variada, a pesar de que en un primer momento podría no parecer tan interesante como otras. Esta variable, además de medir la dificultad de un juego para ser interpretado por personas que tengan algún problema a la hora de leer, efecto que podría modificar la puntuación obtenida por dichos usuarios, realmente nos proporciona cierta información sobre el mercado de una forma muy directa, tal y como veremos a continuación.

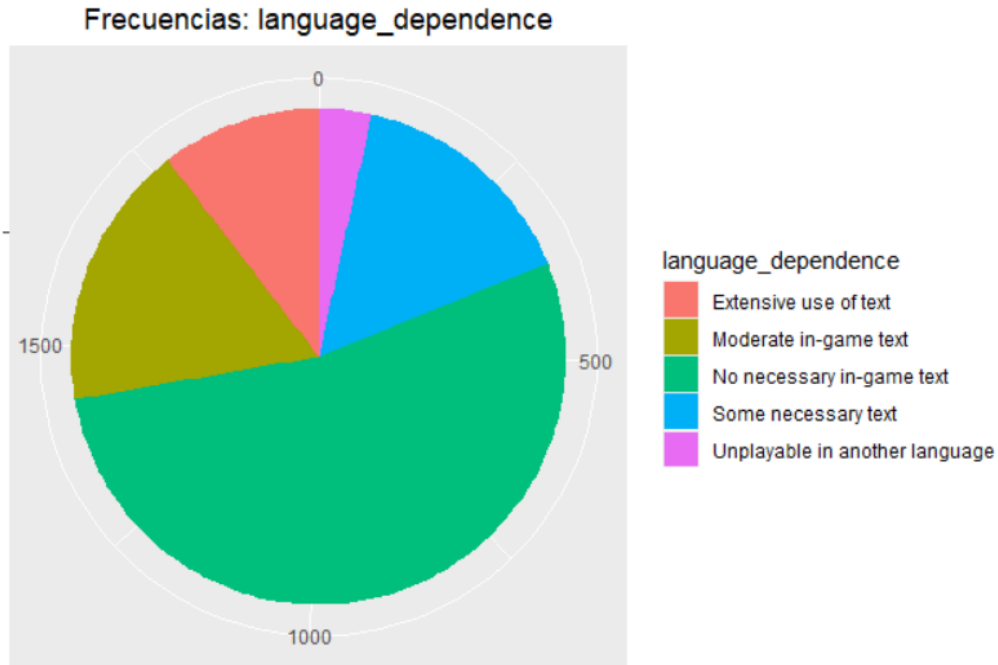

Figura 4.12: Dependencia del lenguaje en los primeros 2720 juegos.

Fuente: Elaboración propia.

Como se puede observar, casi tres cuartas partes de los juegos o bien no incluyen texto vital en el juego o bien solo incluyen un poco, lo cual se podría interpretar como una posible estrategia de mercado, y es que un juego en el que solo haga falta traducir el manual es más barato de producir y exportar, no necesitando la creación de nuevas piezas específicas para ciertos idiomas ni la contratación de un mayor equipo de traductores. Esta dinámica se puede ver, por ejemplo, en los juegos actuales en los que nos encontramos con las reglas en varios idiomas, pero con ninguna palabra luego escrita en pieza alguna, correspondiéndose esto completamente con la idea expuesta. Un buen ejemplo de este comportamiento es el *Colonos de Catán*, nuestro paradigma de juego internacional, como se comentó en el capítulo 1, y es que no incluye prácticamente escritos en sus piezas, siendo de esta forma más sencillo el adaptarlo para su exportación a países de diferente lengua.

Esto se ve reforzado al ver el reducido número de juegos en la categoría “injugable en otro idioma”, muy reducido en comparación con el resto de niveles del factor. Claramente, desde el punto de vista de la producción, esa categoría incrementa los costes, y es por lo tanto algo a tener en cuenta a la hora de diseñar un juego.

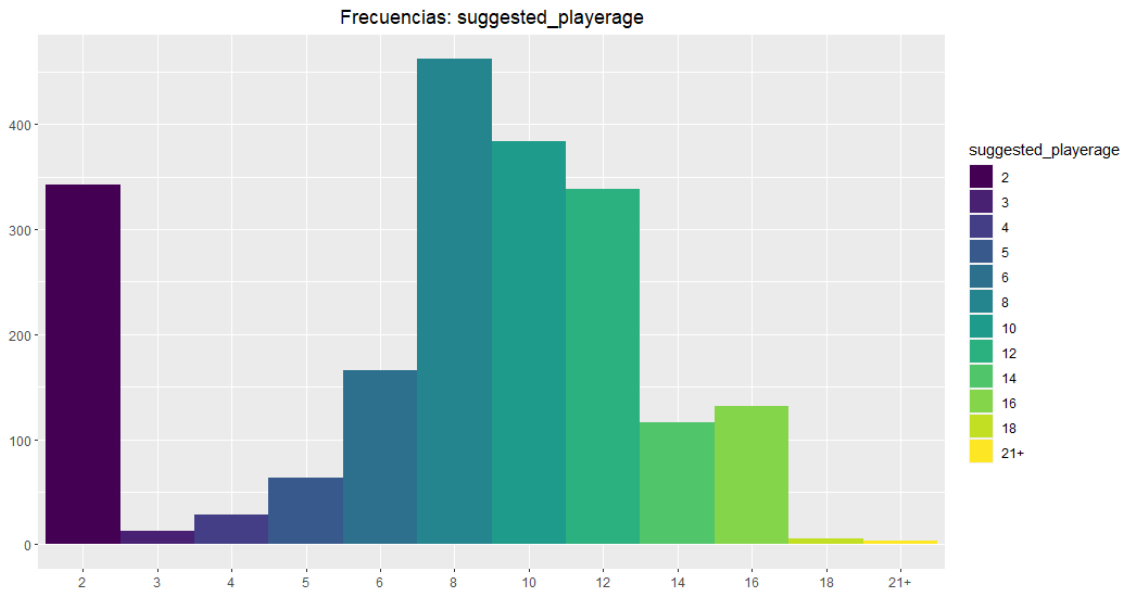

Figura 4.13: Edades mínimas recomendadas en los primeros 2720 juegos.

Fuente: Elaboración propia.

Por otro lado, si centramos nuestra atención en la edad mínima recomendada, *suggested_playerage*, representada en la Figura 4.13, vemos primeramente un

comportamiento aproximadamente normal, con la salvedad del número de apariciones de juegos con una edad mínima de 2 años. En general, parece que esta variable toma, en la mayoría de los casos, valores entre 8 y 12, lo cual nos puede dar una idea, si bien aproximada, del público objetivo de los juegos que se encuentran en la muestra. Sin embargo, es claro que hay un conjunto importante de juegos pensados para los más jóvenes, pudiendo ser interesante, por lo tanto, simplificar esta variable para que tome solo 3 valores: 2 o más años, entre 4 y 8, y más de 8.

Antes de pasar a las siguientes representaciones, en las que nos centraremos en las categorías y en las familias, debemos comentar que, como ya vimos en la Figura 4.4, hay una gran cantidad de familias y categorías posibles, siendo en particular imposible presentar todas las familias en una sola gráfica que aporte información. Es por esto que nos vemos obligados a seleccionar un subconjunto de las categorías para poder obtener gráficos que nos puedan ofrecer alguna guía de cara a futuras secciones. Sin embargo, para reforzar esta idea incluiremos algunas de dichas gráficas.

Empezamos centrándonos en las categorías, construyendo el siguiente histograma para todas las categorías de los primeros 2720 juegos:

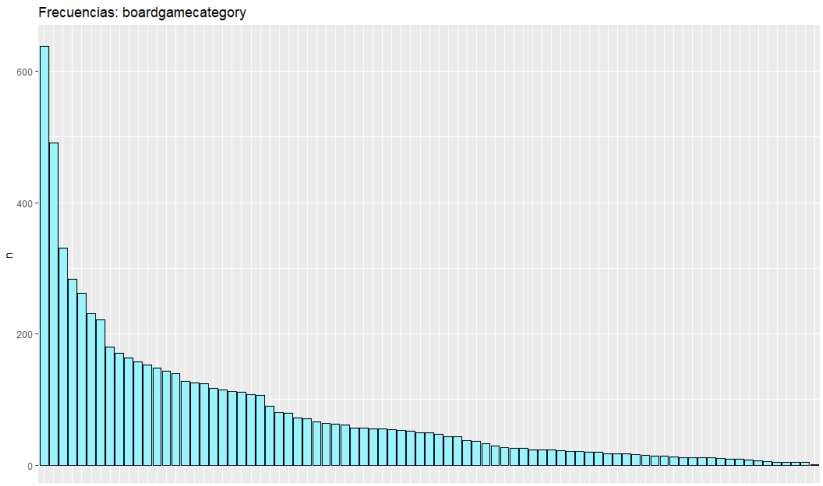

Figura 4.14: Histograma de *boardgamecategory* para los primeros 2720 juegos.

Fuente: Elaboración propia.

En este caso, como mencionábamos, tenemos más de 80 categorías, y observamos un comportamiento de lo más variado, encontrándonos con categorías con menos de 10 entradas y con otras con más de 200, no habiendo un comportamiento general claro.

Intentemos, para obtener un resultado más interpretable, centrarnos en las categorías más frecuentes, que quedan representadas en la siguiente gráfica:

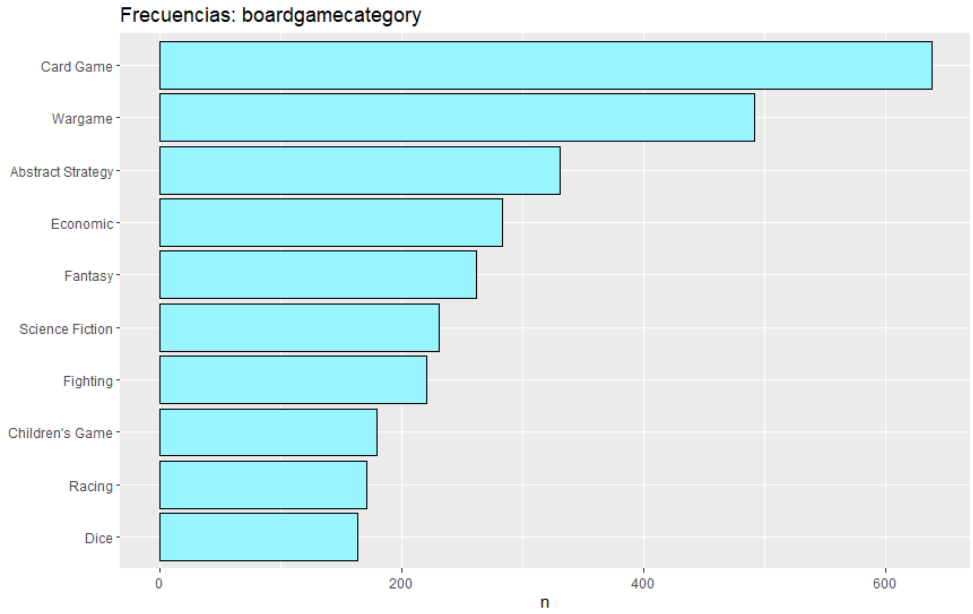

Figura 4.15: Las 10 categorías más frecuentes en los primeros 2720 juegos.

Fuente: Elaboración propia.

En este caso, vemos que son pocas las categorías que superan las 200 entradas, y que se corresponden con temáticas típicas de juegos de mesa: cartas, juego de combate, estrategia, fantasía, economía entendida como gestión de recursos… Este resultado era esperable, pero también nos permite observar, de forma rápida, que el mercado de los juegos de mesa le da especial importancia a esa selección de categorías, centrándose, tal y como era de esperar, en las categorías más clásicas, con las que casi cualquier usuario de juegos de mesa se puede encontrar familiarizado.

Dado que nos centramos únicamente en una variable dentro de los juegos, nos es posible modificar la función *read_bgg* para que lea precisamente estos campos, y ninguno más. Esto nos permite complementar este último gráfico, así como los próximos, con la información extraída para todos los juegos disponibles. Aunque la representación se corresponda en gran medida con la presente en la figura previa, se diferencia considerablemente en el número de entradas nulas, pudiendo entenderse que la BGG asigna categorías a los juegos más antiguos o a los más populares, pues recordemos que tienen almacenados más de 300.000 juegos diferentes.

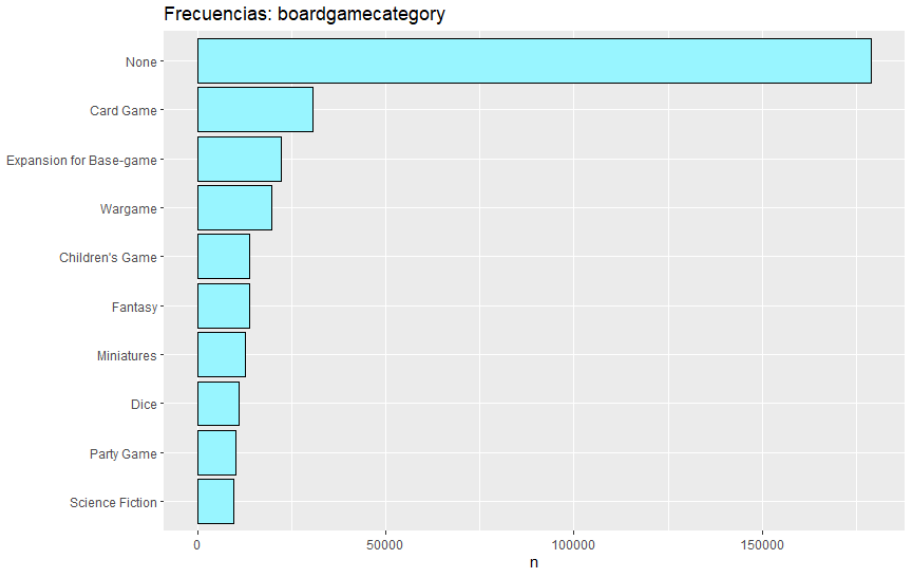

Figura 4.16: Las 10 categorías más frecuentes en todos los juegos leídos.

Fuente: Elaboración propia.

Tal y como mencionábamos, en este caso no observamos grandes cambios, más allá de la incorporación al listado de los juegos más frecuentes de los juegos tipo *party*, incluyéndose en esta categoría juegos como el *Trivial* o el *Time's UP*, más comunes de lo que podríamos haber pensado en un principio. Más adelante, en la sección 6.3, podremos comprobar si estas categorías verdaderamente influyen, por sí solas, en las valoraciones recibidas.

Esperando un resultado semejante al que obtuvimos en las figuras previas, nos centramos ahora en obtener representaciones similares a esta última para las variables *boardgamefamily* y *boardgamemechanic,* siendo esta decisión especialmente acertada al haber más de 1000 categorías posibles para las familias.

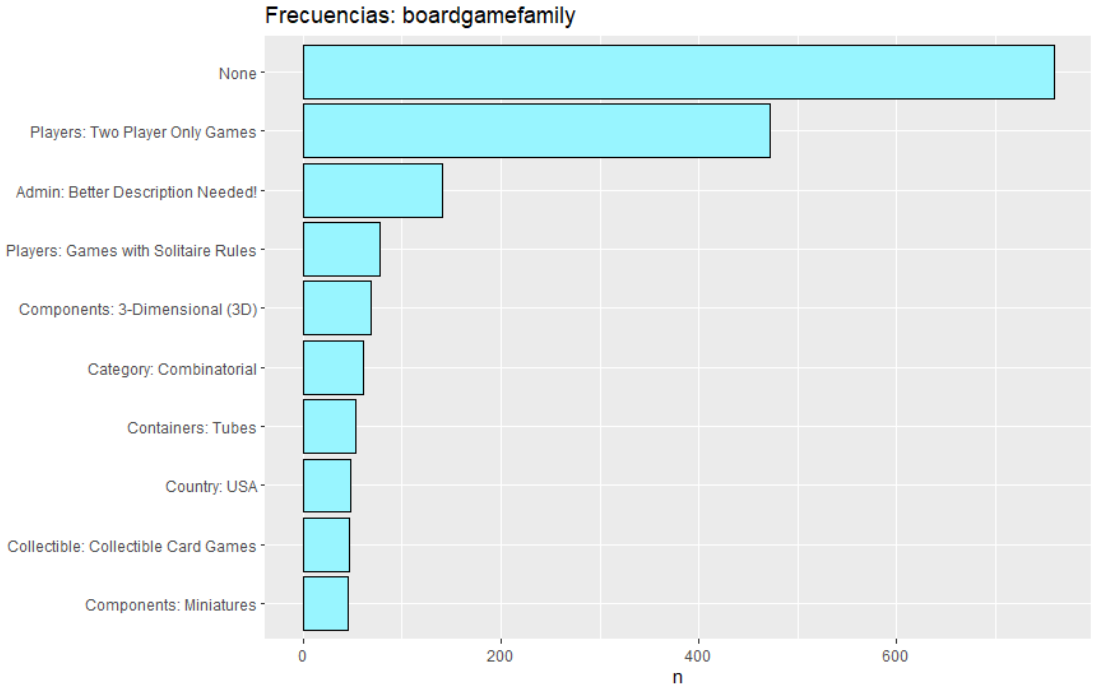

Figura 4.17: Las 10 familias más frecuentes en los primeros 2720 juegos.

Fuente: Elaboración propia.

En este caso comprobamos que no es tan frecuente que un juego tenga una familia asignada, y que realmente incluyen información muy variopinta en esta sección, desde si es un juego pensado únicamente para dos jugadores, a si es estadounidense, o si es necesaria una mejor descripción, siendo esto último una nota para los propios administradores, como ya mencionamos en la sección 3.1.2. Cabe destacar, adicionalmente, que los juegos que admiten una versión en solitario son bastante comunes, al menos como categoría, y se ven seguidos por los juegos con componentes 3D, un campo que hoy en día se encuentra en auge con la aparición de las impresoras 3D. Por otro lado, recordamos que en la API de la BGG no aparece información clara sobre estas familias, dificultando por lo tanto la interpretación de nuestro estudio.

```
bgg_games %>%
  group_by(boardgamecategory) %>%
  count() %>%
  ungroup() %>%
  slice_max(n = 10, order_by = n) %>%
  ggplot(aes(x = reorder(boardgamecategory, n),
             col = I("black"), fill = I("cadetblue1"))) +
  geom_col(aes(y = n)) +
  xlab(NULL) +
  labs(title = "Frecuencias: boardgamecategory") +
  coord_flip()
```

Figura 4.18: Ejemplo del código empleado para la construcción de las figuras de la sección

Fuente: Elaboración propia.

Iteramos como hicimos antes, obteniendo el mismo gráfico pero para todos los juegos disponibles en la base de datos creada, con el objetivo de buscar alguna diferencia apreciable entre la muestra considerada de 2720 juegos y el conjunto global:

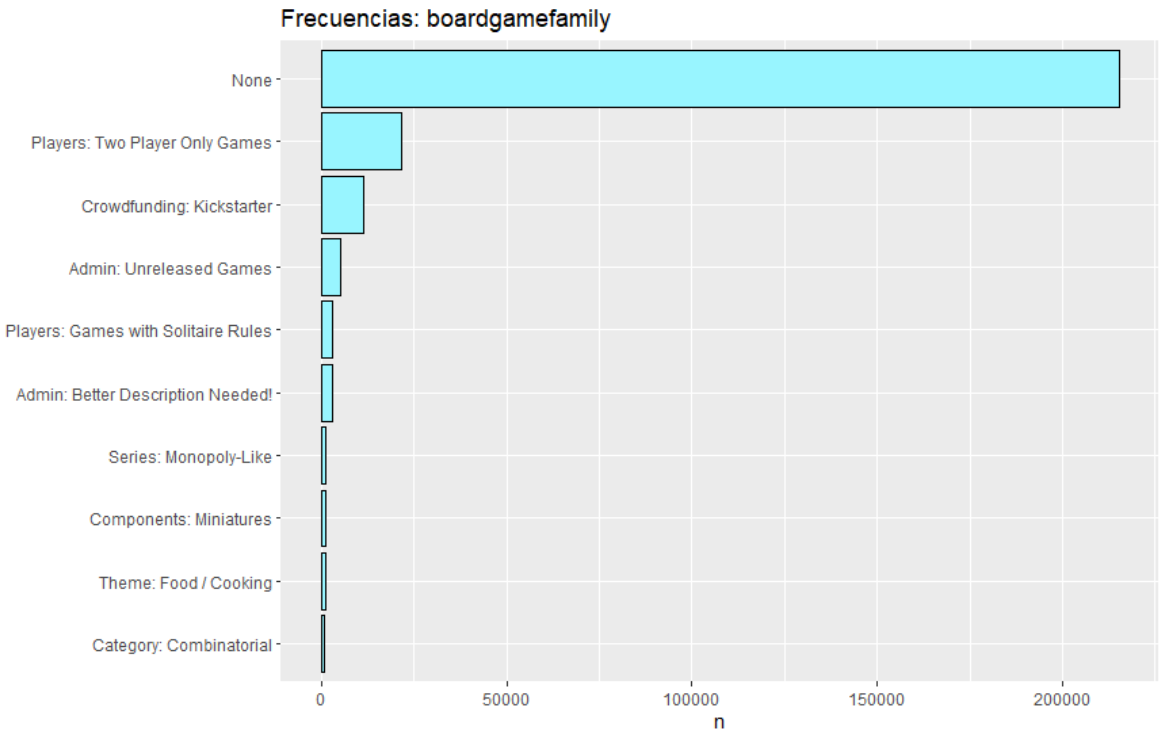

Figura 4.19: 10 familias más frecuentes en todos los juegos leídos.

Fuente: Elaboración propia

.

Vemos que los comportamientos mencionados siguen apareciendo aquí, como los comentarios de los administradores – aunque en esta ocasión sean de dos tipos diferentes –, la ausencia de familia o el número de jugadores, si bien se dispara aún

más la frecuencia de la ausencia. Aparece, también, la familia de los juegos de tipo *Monopoly*, siguiendo esto la línea de lo ya comentado sobre los juegos de tipo *party,* que a la vista de los resultados obtenidos parecen ser más comunes de lo que nos podríamos esperar. Probablemente, juegos de la familia *Monopoly* estén catalogados también como tipo *party*, encontrándonos de esta forma con un problema de dependencia que habría que tener en cuenta más adelante en la modelización, quizás suprimiendo variables y resumiendo la información contenida en ellas, mediante estrategias como el escalado multidimensional (MDS) o el análisis de componentes principales (PCA).

Por último, dentro de esta sección destacamos la aparición de juegos creados mediante mecenazgo en la plataforma *kickstarter.com*, familia que se estudiará más en detalle como serie temporal en la sección 6.4, no solo por ser un gran representante del comportamiento general del mercado, sino porque también la frecuencia de esta familia es considerable, siendo la tercera familia más común, y la segunda si no contamos como familia la ausencia de nivel en esta categoría.

Por último, intentamos repetir lo ya realizado para la variable *boardgamechanic,* con objetivos idénticos a los anteriores.

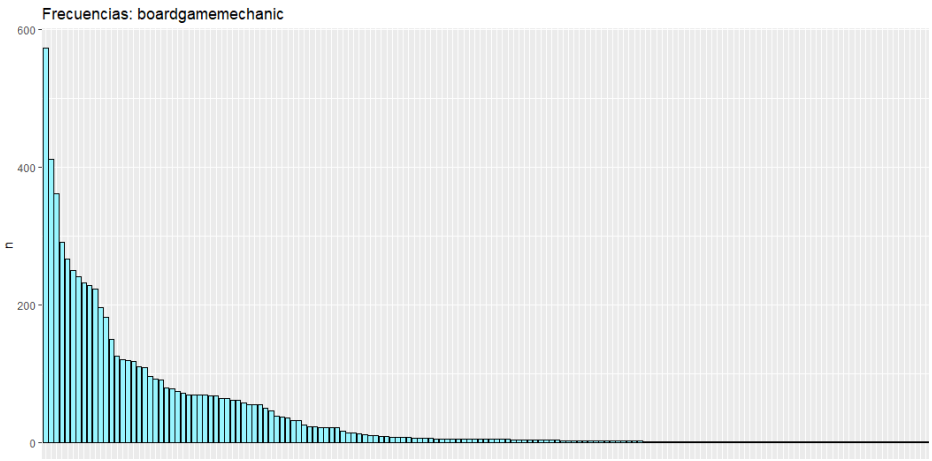

Figura 4.20: Frecuencias de *boardgamemechanic* en los primeros 2720 juegos.

Fuente: Elaboración propia.

Como ocurría con las categorías, aquí las diferencias se marcan aún más, encontrándonos con un reducido conjunto de mecánicas comunes, apareciendo la gran mayoría menos de 10 veces. Veamos, por lo tanto, el comportamiento de las mecánicas más comunes como antes:

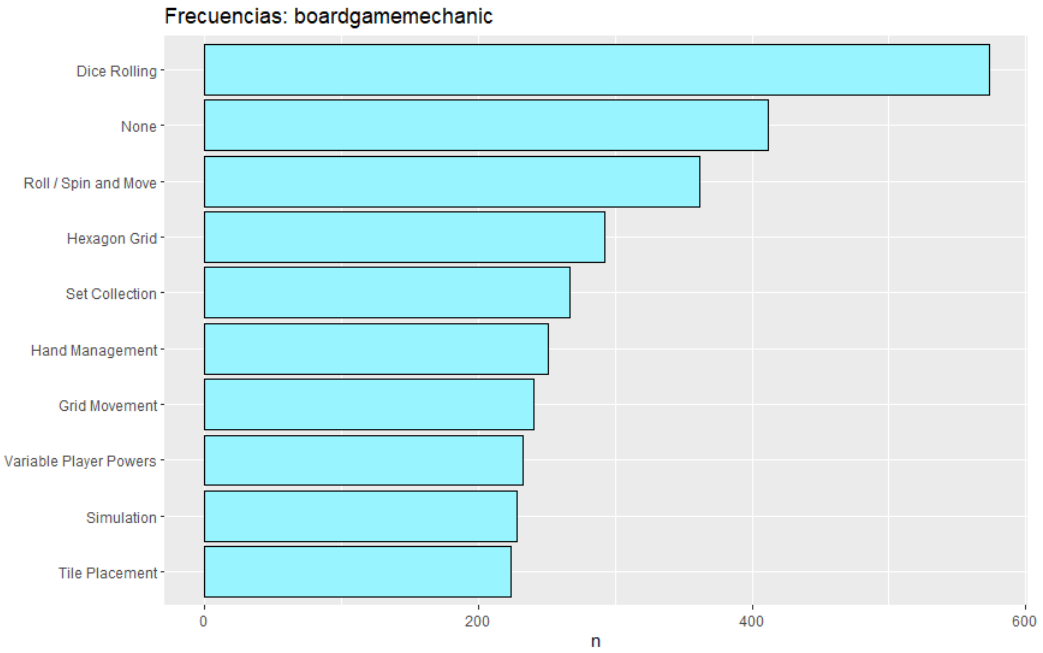

Figura 4.21: Las 10 mecánicas más frecuentes en los primeros 2720 juegos..

Fuente: Elaboración propia.

Como ocurría en los casos previos, la ausencia de categoría es un nivel frecuente en la base de datos consultada, pero eso no nos permite extraer otra información útil de este gráfico, como que las mecánicas más comunes son: el uso de dados, el movimiento determinado de forma aleatoria, el uso de una red hexagonal y la gestión de recursos en mano. Esto es llamativo, pues realmente varias de estas mecánicas se encuentran en los juegos más conocidos como el *Colonos de Catán*, el *Carcassonne* o el *Aventureros al Tren*, siguiendo por lo tanto asumible el pensar que existe un comportamiento generalizado en el mercado por favorecer a este tipo de juegos, aparentemente los más solicitados por el público. Para contrastar rápidamente esta información y ver si es algo propio de la muestra considerada o si representa realmente un comportamiento generalizado, veamos el mismo gráfico pero para todos los juegos:

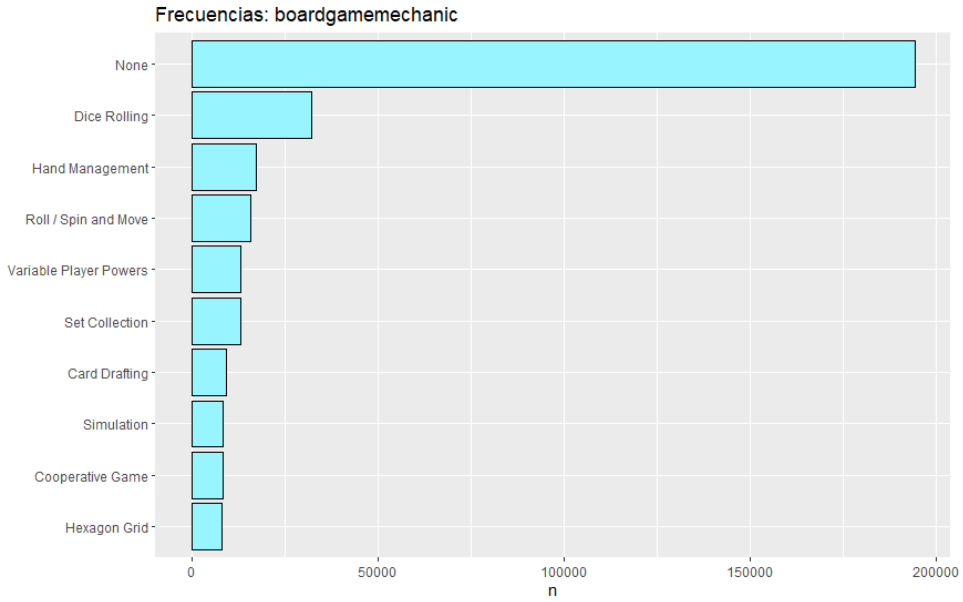

Figura 4.22: Las 10 mecánicas más frecuentes en todos los juegos leídos.

Fuente: Elaboración propia.

Es inmediato observar que solo una o dos mecánicas han cambiado, apareciendo en este caso la mecánica de juegos cooperativos, cada día más comunes. Sí que es importante destacar que el número de *None* se ha incrementado más que en el resto de niveles, pero que, en general, la muestra con la que trabajamos representa bastante bien al conjunto global, al menos en este sentido, compartiendo frecuencias no solo en las mecánicas, sino en las tres categorías consideradas en las últimas páginas.

Por último, para finalizar esta sección cabe destacar la estrecha relación que tienen las mecánicas y familias más frecuentes con las características de los juegos de estilo alemán [29], así como el comportamiento generalizado en lo referido a la dependencia del lenguaje. Esto nos podría animar a realizar un análisis separado, definiendo previamente lo que entendemos como un juego de estilo alemán, agrupando en él diversas mecánicas, familias y categorías.

4.2.2 Series temporales

Buscamos en esta sección estudiar, aunque sea de forma breve, el comportamiento de dos de las variables más interesantes de los datos generados: *average* y *average_weight*, que miden el promedio de la puntuación y de la dificultad de cada juego. Concretamente, analizamos gráficamente la evolución de dichas variables a lo largo del tiempo, con la intención de, más adelante, poder aplicar modelos de series

temporales como el ARIMA para predecir el comportamiento futuro de las mismas, como haremos en la sección 6.4. En este punto, como trabajaremos únicamente con dichas variables, nos es posible trabajar con la totalidad de los juegos descargados en documentos XML, leyendo, con una modificación de *read_bgg,* únicamente los campos de interés, como hicimos en la sección anterior.

Un primer problema con el que nos encontramos a la hora de abordar este punto es el del número de entradas vacías en los datos generados, concretamente en el campo *yearpublished*, pues recordemos que ya vimos que era un campo problemático en la sección 3.1.2. Concretamente, nos vemos obligados a suprimir del estudio en esta parte el 62.11% de los juegos disponibles, quedándonos de esta forma con algo más de 110.000 juegos, lo que sigue pareciendo un número más que aceptable, especialmente si tenemos en cuenta que tenemos entradas para cada año, y no solo eso, sino que también las tenemos para varias categorías, como se verá a continuación.

```
n_na = bgg_games %>%
  filter(is.na(yearpublished)) %>%
  dim %>% extract(1)
N = bgg_games %>%
  dim %>% extract(1)
n_na / N
```

```
## [1] 0.6210769
```

Figura 4.23: Detalle de la proporción de NA en *yearpublished* en los datos descargados.

Fuente: Elaboración propia

.

Además de este punto, una inspección de los datos revela que tenemos entradas hasta para el año -3500 a.C. como ya vimos en la sección 4.1, y estos son datos que no nos aportan información al ser el objetivo de este trabajo el hacer un análisis del sector en la actualidad. Por ello, decidimos suprimir las entradas relativas a juegos previos a 1950, y eliminamos también los de 2020 y 2021, el segundo a causa de que no ha terminado el año a fecha de realización de este proyecto, pudiendo por tanto obtener resultados anómalos; y el primero por una cuestión semejante, si bien en este caso el comportamiento irregular se podría deber a la pandemia de la covid-19.

Por otro lado, para obtener valores que realmente estén validados, para evitar situaciones como la que se nos presentó en el histograma de *average* en la sección anterior, nos restringiremos a aquellos juegos que hayan recibido al menos 50

valoraciones en forma de comentarios y otras tantas puntuaciones de su complejidad, para evitar casos en los que representemos el resultado de una votación de una o dos personas.

En definitiva, seleccionamos un subconjunto de los datos con la siguiente orden:

```
bgg_games %<>%
  filter(yearpublished > 1950 &
         yearpublished < 2020 &
         usersrated > 50 &
         numweights > 50)
```

Figura 4.24: Filtrado de los datos por cuestiones de validez y consistencia.

Fuente: Elaboración propia

Pasando ahora a la representación de la serie temporal, lo primero que ha de llamarnos la atención en la serie de la media de las puntuaciones de los juegos, en la Figura 4.25, es que hemos perdido una gran cantidad de datos, primeramente por la orden de eliminación de entradas con NA, y en segundo lugar por el filtrado al que hemos sometido los datos, representado en la Figura 4.24.

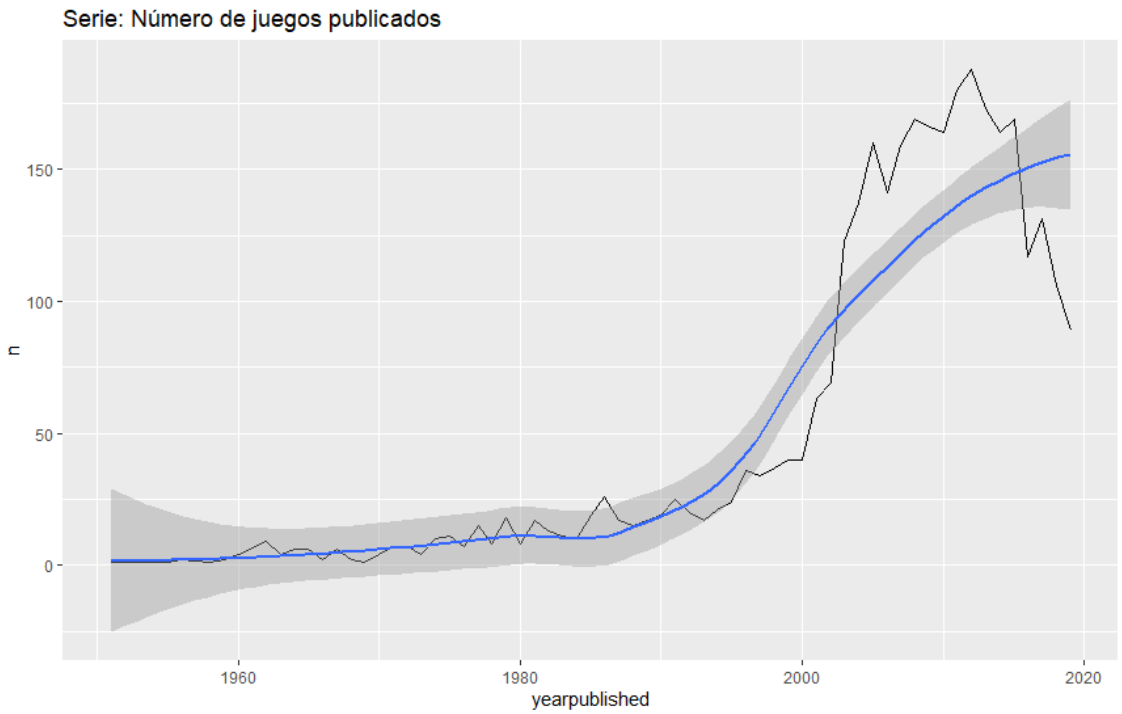

Figura 4.25: Número de juegos publicados con más de 50 valoraciones, 1951-2019.

Fuente: Elaboración propia.

Sin embargo, y esto es lo que nos interesa, podemos extraer de cualquier forma información, pues en los últimos 15 años sí que disponemos de unas 100 entradas o más anuales, lo que nos permite confirmar que los resultados que podamos obtener para dicho intervalo se verán validados por estas frecuencias. Por desgracia, para años anteriores no convendría seguir con el estudio, pues la frecuencia es, en varios casos, totalmente nula.

Buscamos ahora comprobar que este comportamiento, estas frecuencias a lo largo de los años, se conservan anualmente para cada categoría de juego, lo que nos permitirá inferir y predecir más adelante, aplicando los resultados a algunas categorías concretas, como veremos en la sección 6.4. Es por ello que, antes de nada, mostramos en la siguiente figura el número de entradas de cada categoría a lo largo de los últimos 60 años, viendo que, efectivamente, hay categorías para las que no tenemos entradas en determinados intervalos, lo que nos obliga a seleccionar nuevamente un subconjunto de los datos.

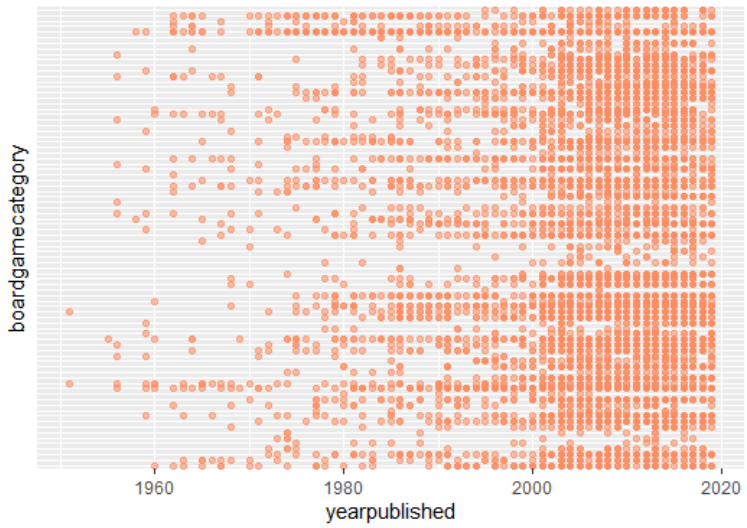

Figura 4.26: Parejas de año-categoría para las que tenemos entradas, 1961-2019.

Fuente: Elaboración propia.

A la vista del gráfico, y combinándolo con lo ya observado en la Figura 4.25, optamos por centrarnos en los años posteriores al 2020, donde parece que disponemos de más entradas. Ejecutamos, por lo tanto, un gráfico como el anterior pero centrado en dicha ventana:

Figura 4.27: Parejas de año-categoría para las que tenemos entradas, 2000-2019.

Fuente: Elaboración propia.

Cabe destacar que estos datos son consistentes, en cierta forma, con la base de datos que hemos consultado, pues al haberse fundado la BGG en el año 2000, es esperable que tengan un mayor número de entradas para juegos desde ese año, siendo necesaria cierta investigación para poder incluir información semejante para juegos previos.

Terminamos esta sección tomando el subconjunto de categorías para las que hay entradas cada año, entre el 2000 y el 2019, y construyendo varios gráficos sobre estas series, obteniendo un resultado particular, como se puede ver en la siguiente figura:

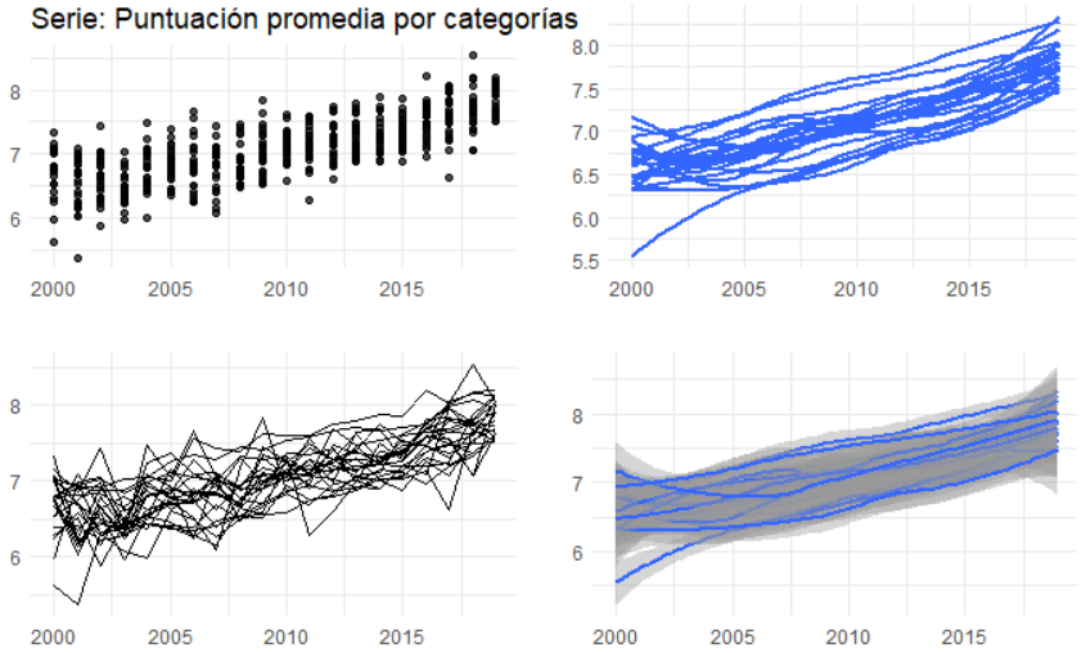

Figura 4.28: Parejas año-puntuación por categoría, incluyendo regresión loess e intervalo de confianza.

Fuente: Elaboración propia.

A la vista de estas gráficas, podríamos plantearnos la hipótesis de que la puntuación de los juegos, de las categorías más comunes, se ha ido incrementando en los últimos años, esto es, los usuarios están cada vez más satisfechos con los juegos que se lanzan al mercado dentro de las categorías más habituales. Esta idea la tendremos en mente en el capítulo 6, donde intentaremos modelizar el comportamiento de estas puntuaciones, y concretamente a lo largo del tiempo, recurriendo a modelos ARIMA, entre otros.

4.2.3 Puntuación por categoría

Dado que queremos intentar discernir las claves del éxito dentro del sector, esto es, qué variables hay que controlar para obtener la mayor puntuación posible, estudiamos las diferentes puntuaciones por categoría, pudiendo ser esta una fuente de variación controlable dentro de la puntuación.

Sin embargo, como ya vimos en la Figura 4.4, disponemos de unas 80 categorías solo en los primeros 2720 juegos, lo que nos obliga a hacer una selección de las categorías para poder interpretar estos gráficos.

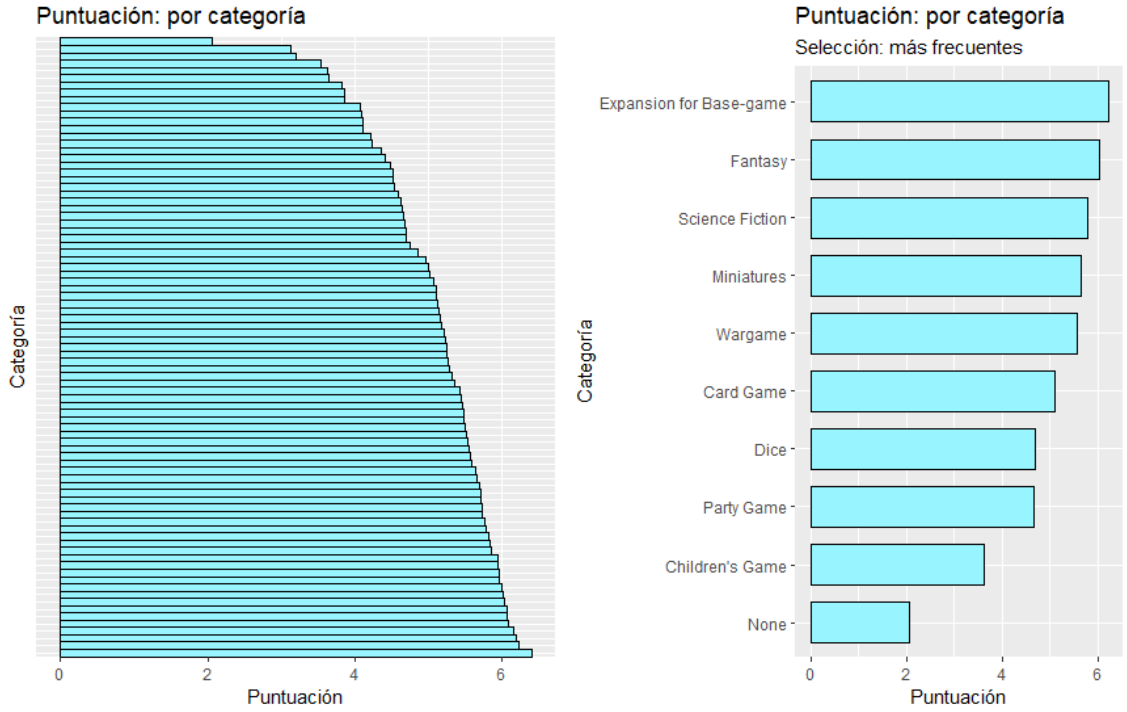

Figura 4.29: Puntuación por categoría para todos y para las categorías más frecuentes en todos los juegos.

Fuente: Elaboración propia.

De esta forma, vemos en la Figura 4.29 que el promedio de las puntuaciones varía entre algo más de un 2 y poco más de un 6, luego primeramente resultaría plausible el considerar la categoría como una fuente de variabilidad en la variable *average*, si bien habremos de comprobar esto más adelante, en la sección 6.3. Cabe destacar, asimismo, que las puntuaciones de las categorías más frecuentes son muy próximas entre sí, lo que tendría hasta cierto punto sentido, si lo interpretamos como que las editoriales intentan producir juegos de temáticas que sepan que ya han triunfado, como los juegos de miniaturas, de fantasía o de ciencia ficción.

Sin embargo, nos llama la atención ver cómo las categorías *Children's Game* y *None* tienen asociadas puntuaciones bajas, pudiendo deberse en el caso del primero a que la comunidad de la BGG es mayoritariamente adulta, y el segundo a que juegos sin categoría o bien son demasiado diferentes a todo lo que el público puede conocer y por lo tanto reciben puntuaciones bajas, o bien existe otra justificación más sencilla, y es que los administradores de la BGG optan por asignarle categorías únicamente a los juegos más exitosos por cuestiones logísticas, esperando que los usuarios no busquen dicha información en un juego poco popular.

Con estos análisis hemos logrado intuir algunos puntos del comportamiento interno de los datos seleccionados, lo que nos anima a construir modelos de regresión para intentar explicar los comportamientos vistos hasta aquí, punto que se abordará en el capítulo 6.

5 PROCESAMIENTO DEL LENGUAJE NATURAL

Tal y como ya comentamos, este campo dentro de la inteligencia artificial no ha sido prácticamente estudiado a lo largo del currículo académico, y es por esta razón que le dedicamos un capítulo entero, siendo el objetivo principal del mismo el intentar aplicar las herramientas explicadas en el capítulo 2 para múltiples cuestiones, desde obtener una semejanza entre juegos a un análisis de sentimiento, pasando por el modelo de ratios.

En particular, nuestra atención se centrará en la relación entre los comentarios y las puntuaciones asociadas a los mismos, o la puntuación del juego, pues recordemos que es la medida que hemos tomado como importancia del juego dentro del mercado.

5.1 Aplicación del modelo vectorial

Ya explicamos en la sección 2.5.2 la fundamentación del modelo, y es en este apartado donde intentaremos aplicarlo, recurriendo primeramente a diferentes gráficos y tablas para ver su aspecto y, posteriormente, poder emplearlo para un doble propósito: para ver si con la representación vectorial de un modelo somos capaces de predecir la puntuación asociada al comentario, lo que sería una utilidad clara, y para aprovechar la función de similitud que ya definimos en la sección mencionada, abordando primero la tarea de ver la similitud entre comentarios de un mismo juego, para luego generalizar y obtener una funcionalidad que nos proporcione, dado un juego, aquellos que según los comentarios son más semejantes.

5.1.1 Análisis de los comentarios con el modelo de espacio vectorial

Buscamos ahora en esta sección aplicar el modelo de espacio vectorial, explicado en la sección 2.5.2, de forma tal que podamos obtener conclusiones sobre la similitud entre diferentes documentos, pudiéndose emplear estar información para predecir comentarios, como en el modelo KNN; además, también construiremos una función que nos proporcione los juegos más semejantes a uno dado, basándonos únicamente en el contenido de los comentarios.

Ya introdujimos en el capítulo 2 los paquetes `tm` y `tidytext`, y es aquí donde realizaremos un uso extensivo de ellos, empleándolos en cada paso que demos. Por otro lado, por cuestiones computacionales, nos restringiremos a un subconjunto de los juegos generados en estos casos, dado que, como ya vimos en el subcapítulo 4.1, de análisis descriptivo, podemos contar con más de 18.000 comentarios para ciertos

juegos. Sin embargo, la generalización de las ideas y modelos aquí explicados es inmediata, bastando con incrementar el tamaño muestral.

En definitiva, trabajaremos con una variación de nuestro objeto *bgg_games*, en la que tomaremos los 10 primeros documentos *games_i.xml*, suponiendo esto unos 3.000 aproximadamente, y trabajando con una muestra de 500 comentarios, dado que en muchos casos estos pueden o bien estar vacíos o tener una puntuación no declarada.

Antes de nada, como comentamos en la sección 2.2.3 sobre el paquete `tm`, para poder trabajar con él es necesario tener preparada una carpeta en la que dispongamos de todos los documentos con los que vayamos a trabajar, razón por la cual diseñamos, a partir del objeto *bgg_games*, una función que admita como parámetro de entrada el número de juegos que escribir y que cree la carpeta *game_i*, donde almacenará cada comentario por separado. El resultado es el siguiente, para $n = 3$:

Figura 5.1: Carpetas de trabajo para el paquete `tm`.

Fuente: Elaboración propia.

Ahora trabajaremos con el primer juego, guardado en *game_1*, a modo de ejemplo, pudiéndose extender las funcionalidades generadas para cualquier juego con facilidad, simplemente cambiando la carpeta de trabajo.

Antes de nada cargamos con la orden *VCorpus* el corpus volátil, la estructura con la que trabaja el paquete `tm`, que nos indica únicamente los metadatos y el número de documentos leídos. El adjetivo volátil se debe a que, tal y como indica nuestra referencia [2], el objeto desaparece si lo hace también la carpeta contenedora.

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 500
```

Figura 5.2: Respuesta por consola al cargar un corpus con *VCorpus*.

Fuente: Elaboración propia.

Aprovechamos que hemos escrito los juegos de forma ordenada y clara – escribiendo ceros hasta rellenar las 3 cifras, como vimos en la Figura 5.1 –, y que `tm` nos permite asignar metadatos con facilidad pero en orden, para guardar junto con cada documento, cada comentario, la puntuación asociada al mismo, pudiendo ser esta ocasionalmente NA.

A continuación, para poder seguir necesitamos depurar el texto, pues antes de separar en palabras deberíamos intentar suprimir palabras de poco interés, como los números, los conocidas *stopwords*, generalmente conectores u otras partículas que no aportan directamente información, así como los espacios innecesarios, pasando en el proceso todo a minúscula. Todo esto se puede hacer con rapidez con la función *tm_map*, que nos permite aplicarle a cada documento de un corpus volátil una de las utilidades del paquete, que incluyen, entre otras, las que hemos comentado.

Concretamente, para ejecutar lo indicado basta con hacer:

```
game_1_corpus %<>%
  tm_map(stripWhitespace) %>%
  tm_map(content_transformer(tolower)) %>%
  tm_map(removeWords, stopwords("english"))) %>%
  tm_map(removePunctuation) %>%
  tm_map(removeNumbers)
```

Figura 5.3: Limpieza y guardado del corpus *game_1_corpus* con el paquete `tm`.

Fuente: Elaboración propia.

Adicionalmente, aprovechamos para tomar una decisión, y es que asumiremos que todos los comentarios están en inglés, pues el hecho de que la plataforma use el inglés como lenguaje ya condiciona en gran medida el idioma escogido por los usuarios a la hora de comunicar sus opiniones. Un vistazo rápido a varios conjuntos de comentarios apoya esta decisión, y es por ello que a la hora de quitar las *stopwords* solo suprimimos las que están en inglés. Para ver un análisis algo más detallado de esta decisión se puede acudir a la sección 7.2.

Siguiendo ahora con nuestro modelo vectorial, necesitamos una matriz de frecuencias, que indique, para cada pareja de documento D y término t, el número de apariciones de t en D, esto es, $\mathrm{tf}_{t,D}$, como la definimos en la sección 2.5.2, con el objetivo de obtener las otras frecuencias que son necesarias para construir la representación vectorial de cada documento en este modelo.

Ahora bien, el paquete `tm` incluye las funciones *DocumentTermMatrix* y *TermDocumentMatrix*, que tal y como nos podríamos imaginar devuelven las matrices documento-término y término-documento, siendo una la traspuesta de la otra.

```
## <<DocumentTermMatrix (documents: 500, terms: 2786)>>
## Non-/sparse entries: 6639/1386361
## Sparsity           : 100%
## Maximal term length: 138
## Weighting          : term frequency (tf)
## Sample             :
##               Terms
## Docs           can fun game games good like one play played players
##   text_079.txt   4   0    3     1    0    1   0    1      0       2
##   text_101.txt   0   0    0     0    0    0   0    0      0       0
##   text_134.txt   0   1    4     1    0    2   2    3      0       1
##   text_163.txt   1   0    4     2    0    0   3    1      0       0
##   text_237.txt   0   0    3     0    3    1   0    0      2       0
##   text_240.txt   1   0   14     4    1    1   2    1      0      10
##   text_367.txt   0   0    3     3    1    0   1    0      2       0
##   text_376.txt   2   0    9     1    0    2   0    4      0       0
##   text_448.txt   1   2   10     0    2    0   1    1      0       3
##   text_479.txt   1   1    5     0    0    1   1    1      0       2
```

Figura 5.4: Resultado de la inspección de la matriz documento-término asociada a nuestro corpus.

Fuente: Elaboración propia

Desgraciadamente, la salida que nos da, aunque resulte clara con la orden *inspect* como observamos en la Figura 5.4, tiene un problema destacable, y es que nos devuelve un objeto *simple triplet matrix* del paquete `slam` [30], una matriz que está guardada como un triplete (fila, columna, valor), lo cual no sirve del todo a nuestros propósitos y nos obliga a crear una función que transforme esta información en un *tibble*, estructura que intentaremos usar siempre que podamos por las herramientas que estamos empleando.

Una vez hecho esto, generamos los vectores de frecuencia documental y frecuencia documental inversa, representándolos en histogramas para ver qué distribución parecen seguir:

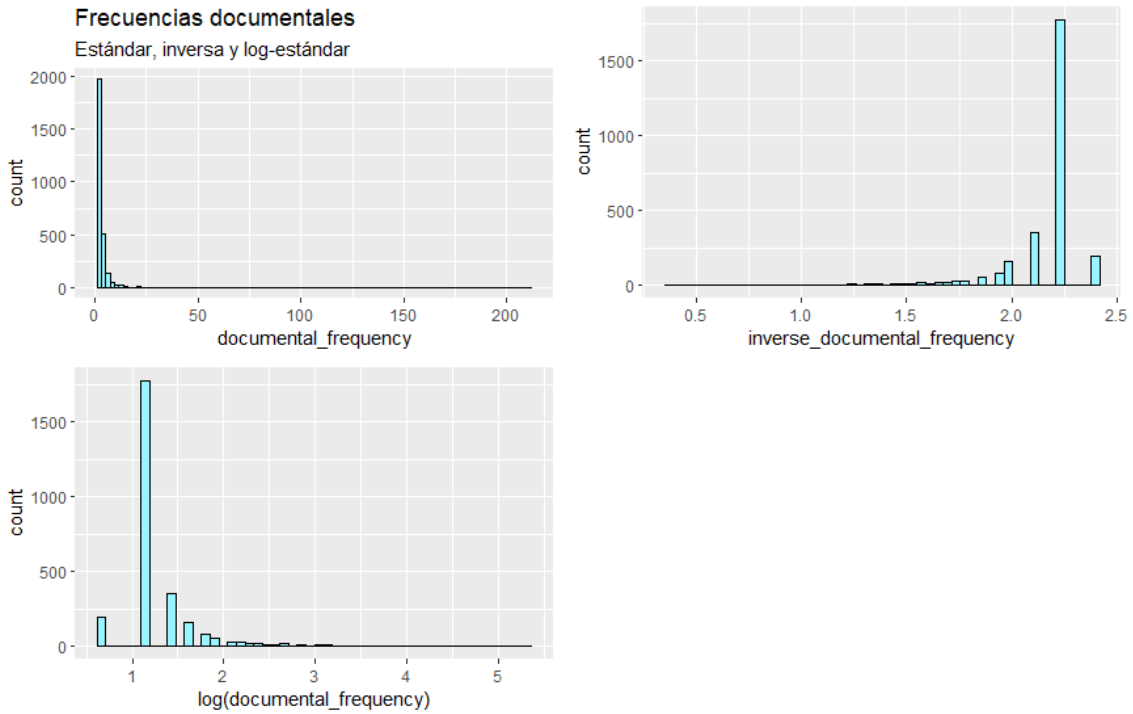

Figura 5.5: Histogramas de la frecuencia documental, frecuencia documental inversa y log-frecuencia documental.

Fuente: Elaboración propia

Observamos que las frecuencias documentales se encuentran concentradas alrededor de valores relativamente bajos, casi todos menores a 25, y recordemos que estamos trabajando con 500 comentarios para este juego, lo que es un resultado esperable, pues no es común que un elevado número de palabras se repita a lo largo de un gran conjunto de comentarios, más allá de las propias del contexto, como "*game*" o el propio nombre del juego. Por otro lado, las frecuencias inversas se encuentran concentradas alrededor de valores entre 2 y 2.5, lo que favorecerá a nuestro modelo, pues a partir de ella construimos los pesos y la posterior representación vectorial, que a su vez nos asegurará cierto comportamiento conveniente más adelante, con la función de similitud, en la siguiente sección.

Añadidos estos valores al *tibble* de frecuencias que habíamos generado a partir de la matriz dada por la orden *DocumentTermMatrix*, pasamos a realizar el cálculo principal del modelo vectorial: la obtención de los pesos de cada palabra en el documento. Sin embargo, antes de ofrecer resultados, debería llamarnos la atención la aparición en la matriz de la Figura 5.4 de multitud de palabras comunes a todos los juegos de mesa,

como “*players”*, “*play”* o “*game”*, y que convendría suprimir del vocabulario al no aportar información alguna sobre la semejanza entre diferentes documentos.

Recordando ahora que en la sección 2.5.2 vimos la importancia de seleccionar un vocabulario conveniente, decidimos, a modo de guía, recurrir al paquete `wordcloud`, para ver cuáles son las palabras más comunes en este juego, de forma tal que obtengamos una primera aproximación al vocabulario, viendo adicionalmente qué palabras deberíamos suprimir al ser demasiado comunes dentro del sector de estudio.

```
wordcloud(words = freq_tibble$term,
          freq = freq_tibble$documental_frequency,
          max.words = 100,
          colors = brewer.pal(n = 5, name = "Set2"))
```

Figura 5.6: Orden para generar la nube de palabras con el paquete `wordcloud`.

Fuente: Elaboración propia

Con la anterior orden, dado el *tibble* con los términos y las frecuencias documentales generadas en los anteriores pasos, construimos la nube de palabras con un máximo de 100 palabras, siendo el tamaño de cada palabra proporcional a su frecuencia, dependiendo también tanto el color como la posición, más o menos cercana al centro, dependientes de esta frecuencia. Adicionalmente, con la utilidad *brewer.pal*, del paquete `RColorBrewer`, obtenemos una paleta de 5 colores con los que realizar la representación, seleccionando el conjunto *Set2* al estar catalogado por este paquete como apto para daltónicos. Aprovechando esta utilidad, a lo largo del trabajo intentaremos emplear, siempre que sea posible, colores seleccionados dentro de esta gama.

Figura 5.7: Nube de palabras de los comentarios del primer juego, basada en las frecuencias documentales.

Fuente: Elaboración propia

Nótense en esta nube de palabras un par de detalles: el primero, que esta es una funcionalidad que realmente podría incluirse en la página, que computacionalmente no cuesta demasiado y que resume de una forma visualmente atractiva el contenido de muchísimos comentarios; el segundo, que nos permite reafirmar la decisión que tomamos hace unos párrafos: la asunción de que la inmensa mayoría de los comentarios estaba en inglés.

Esta nube de palabras refleja con rapidez el comportamiento general solo con la frecuencia documental, y si bien podríamos hacer un gráfico semejante para la frecuencia inversa, el resultado no es de sencilla interpretación, y aclara poco, razón por la cual no incluimos dicha gráfica en este punto.

Volviendo a la que era nuestra labor en este momento, la selección de un vocabulario adecuado, recurrimos a la función *get_sentiment* del paquete `tidytext`, que nos proporciona diferentes conjuntos de sentimientos con los que trabajar: *bing*, *nrc* y *afinn*. Dado que su uso se centra principalmente en el análisis de sentimiento, dejamos a un lado la justificación y la explicación de los mismos hasta la siguiente sección, asumiendo por ahora que son diccionarios de una longitud "adecuada", rondando las 6.000 palabras en cada caso. Por lo tanto, para seleccionar un vocabulario que aporte información suprimimos del *tibble* de frecuencias las palabras relativas a los juegos de mesa de las

que hablamos hace poco, y al resultado le aplicamos un *inner_join* con el diccionario *bing*, de forma tal que nos quedamos únicamente con el 15% de las palabras originales, 434 de las 2786 de las que partíamos. Esto facilitaría la interpretación del modelo, y nos asegurará coherencia en los resultados, pero podríamos perder potencia, razón por la cual seleccionamos también el 10% de las palabras con una mayor frecuencia documental y con una mayor frecuencia documental inversa, añadiéndoles a estas las 434 palabras que ya teníamos, pudiéndose repetir en este paso algunas.

En definitiva, logramos construir un vocabulario con 900 palabras, que nos permitirá darle forma a representaciones de difícil interpretación pero que nos asegurará hasta cierto nivel una determinada potencia en el modelo. Sin embargo, como hemos tomado un vocabulario tan extenso, si queremos ver el número de elementos no nulos por columnas en nuestra representación vectorial obtenemos:

Figura 5.8: Histograma del número de entradas no nulas por palabra, esto es, la frecuencia documental para nuestra selección.

Fuente: Elaboración propia

Tenemos una concentración importante de palabras que aparecen únicamente en un documento, pero la mitad parece haber conservado unas frecuencias más aceptables para nuestros intereses, incluyéndose algunas con una frecuencia documental más

reducida, y que por lo tanto podemos entender que su selección se debe al haber tomado los elementos con mayor frecuencia documental inversa también.

Siguiendo con el modelo, construimos ya la matriz de pesos con este vocabulario, esto es, representación vectorial, obteniendo representaciones para cada documento del corpus como la que siguen, si bien con 900 filas por la longitud de nuestro vocabulario:

term <chr>	text_001.txt <dbl>
feels	1.920819
fun	0.000000
makes	1.522879
silly	0.000000
stage	2.221849
time	1.337242

Figura 5.9: Fragmento de la representación vectorial de *text_001.txt*.

Fuente: Elaboración propia

Aprovechando esta representación, queremos intentar construir ahora un modelo para predecir la puntuación de un comentario dada su representación vectorial tomando, para cada comentario, el peso de cada palabra como variable predictora. Una vez tengamos esto, aplicaremos KNN y árboles de regresión, modelos estudiados a lo largo del grado y que no nos imponen prácticamente condiciones para su uso. Recuperando los metadatos que almacenamos en el corpus, la puntación de cada comentario, y gracias a los nombres que les hemos dado a las columnas, logramos construir una matriz que, previa eliminación de los NA y trasposición de la matriz de la representación, tiene en cada fila la puntuación del comentario y la representación del mismo, apareciendo en las columnas la puntuación y las palabras de nuestro vocabulario.

Implementamos ahora KNN y árboles de regresión siguiendo la filosofía del paquete `tidymodels`, empezando por lo tanto con una división en conjuntos de entrenamiento y test, *dtm*_split, para poder predecir, y la complementamos con un conjunto con 10 pliegues sobre los mismos datos de entrenamientos para estimar los diferentes errores mediante validación cruzada. Notemos que le hemos aplicado *make.names* a las columnas de *dtm_tibble*, pues trabajamos con caracteres que podrían ser incompatibles con algunas de las funciones consideradas. A continuación, como podemos observar en la Figura 5.10, definimos una receta en la que normalizamos los datos numéricos de entrada, pasamos un filtro de varianza con la función *step_zv*, eliminando de esta forma columnas constantes si las hubiera, y finalmente omitimos las posibles filas con NA,

ejecutando este paso también con el conjunto tipo test, si bien no deberíamos generar en ningún caso valores de este tipo.

```
colnames(dtm_tibble) %<>% make.names
dtm_split = initial_split(dtm_tibble)
dtm_tibble_cv = cv_folds = vfold_cv(training(dtm_split), v = 10, breaks = 4)

my_rec = recipe(comment_rating ~ .,
                data = training(dtm_split)) %>%
  step_center(all_numeric()) %>%
  step_scale(all_numeric()) %>%
  step_zv(everything()) %>%
  step_naomit(everything(), skip = FALSE)

my_wf = workflow() %>%
  add_recipe(my_rec)
```

Figura 5.10: Definición de los pliegues, de la receta y del flujo de trabajo.

Fuente: Elaboración propia

Siguiendo con la filosofía de `tidymodels`, incluimos la receta creada a nuestro flujo de trabajo, *my_wf*, listo ahora para admitir modelos y conjuntos de datos con los que trabajar. Destacamos en este punto la importancia de la normalización de los datos al no tener garantía a priori de que todas las entradas tengan los mismos órdenes de magnitud, hecho que podría corromper el modelo, reduciendo su eficacia.

Una vez tenemos nuestro objeto *workflow*, definimos con la ayuda del paquete `parsnip` los modelos KNN en la Figura 5.11 y el modelo de árboles de regresión en la Figura 5.12. Añadiendo este modelo al flujo de trabajo y con la ayuda de la orden *fit_resamples,* empleando como conjunto de remuestreo el ya definido, con 10 pliegues, obtenemos las métricas asociadas por defecto, el error cuadrático medio y el R^2.

```
knn_spec = nearest_neighbor() %>%
  set_engine("kknn") %>%
  set_mode("regression")

my_wf %>%
  add_model(knn_spec) %>%
  fit_resamples(resamples = dtm_tibble_cv,
                metrics = metric_set(rmse, rsq)) %>%
  collect_metrics() %>% select(.metric, mean)
```

.metric <chr>	mean <dbl>
rmse	0.98179357
rsq	0.07706661

Figura 5.11: Definición del modelo KNN y obtención de las métricas mediante validación cruzada con 10 pliegues.

Fuente: Elaboración propia

Con esto, aunque en otras condiciones podríamos decir que el modelo es correcto al haber obtenido un error cuadrático medio de algo menos de 1, en nuestro caso nos indica que el modelo no tiene prácticamente validez, pues recordemos que estamos trabajando con datos normalizados, que en casi toda ocasión tomarán valores entre -1 y 1. Esto explica en parte el reducido R^2 obtenido, y es que nuestro ajuste explica muy poco el comportamiento de las puntuaciones, siendo por lo tanto descartable como modelo apropiado para esta situación.

Sin embargo, podríamos encontrarnos con un resultado diferente a la hora de aplicar árboles de regresión al tener estos una fundamentación muy diferente, basada en centrarse en los elementos más dispares con mayor variabilidad en lugar de en aquellos más cercanos, pudiendo obtener, por lo tanto, valores diferentes.

```
rt_spec <-
  decision_tree() %>%
  set_engine(engine = "rpart") %>%
  set_mode("regression")

my_wf %>%
  add_model(rt_spec) %>%
  fit_resamples(resamples = dtm_tibble_cv,
                metrics = metric_set(rmse, rsq)) %>%
  collect_metrics() %>% select(.metric, mean)
```

.metric <chr>	mean <dbl>
rmse	0.98429297
rsq	0.09374463

Figura 5.12: Definición del modelo de árboles de regresión y obtención de las métricas mediante validación cruzada con 10 pliegues.

Fuente: Elaboración propia

En este caso ocurre lo mismo que en el modelo anterior, y es que cometemos un error semejante al que obtendríamos de asignar valoraciones de forma aleatoria, si bien se puede observar una mejora en el R^2, aunque sea reducida.

Esto nos permite afirmar que, al menos con los modelos aquí expuestos, la representación vectorial de cada documento no sirve como conjunto de variables con las que predecir la puntuación asociada al comentario, o al menos no en general, pudiendo quizás mejorar los resultados aplicando otros procedimientos más avanzados de procesamiento del lenguaje natural, como los que se vieron en la sección 2.5.3.

5.1.2 Similitud en el modelo vectorial

Recordemos que ya habíamos definido la similitud entre dos documentos como:

$$sim(\overline{D}, \overline{D'}) = \frac{\sum_{i=1}^{r} d_i d_i'}{\sqrt{\sum_{i=1}^{r} d_i^2} \sqrt{\sum_{i=1}^{r} d_i'^2}}$$

Así, definimos ahora una función que, dados dos vectores, calcule su similitud, teniendo en cuenta que es 1 si los dos son nulos y que no se puede calcular si solo una es nula. Por ejemplo, comparando los comentarios 103 y 105 del primer juego obtenemos una similitud de 0.56, aproximadamente, cuando dichos comentarios eran:

- *Fun and interesting game. One of the first german game I picked up. Very colourful and lots of different paths to victory.*
- *Super-quick, easy to teach, and interesting psych-out games lie within.*

Parece que la semejanza se debe a la aparición de "*interesting*", si bien no podemos comentar más en un primer momento.

Ahora, para poder explotar al máximo este modelo, construimos una función, *query_to_representation*, que convierte una consulta de texto en una representación con el formato de la matriz de representaciones. Obtenemos, por ejemplo, para la consulta "*It was a really good game. Great opponent, good match*", la siguiente representación:

term <chr>	text_rep <dbl>
fun	0.000000
one	0.000000
like	0.000000
good	4.641572
can	0.000000
really	2.474936
much	0.000000
rules	0.000000
little	0.000000
great	2.746870

Figura 5.13: Ejemplo de representación vectorial para una consulta dada. Columnas seleccionadas.

Fuente: Elaboración propia

Tal y como era de esperar, muchas de las entradas son nulas al ser un documento breve, pero igualmente disponemos de varias no nulas, que nos permitirán ver la similitud de este comentario con otros. Aplicando esto, podemos sustituir, por ejemplo, valoraciones perdidas por la valoración del comentario más semejante según esta similitud, siendo esto realmente un modelo KNN con $k = 1$. Otra aplicación sería la de seleccionar comentarios semejantes a uno dado, para presentar aquellos que traten el mismo tema, que tengan el mismo enfoque o que cumplan, en general, cierta propiedad de interés para el analista.

Una vez hemos visto la idea con un juego, intentamos ir directamente más allá, trabajando con la similitud entre **diferentes juegos**, dados unos nuevos corpus que habremos de construir para poder proceder como hemos hecho hasta ahora.

Nuevamente debemos escribir los comentarios de un formato apto para el paquete `tm`, siendo el resultado una carpeta con 300 documentos, uno asociado a cada uno de los primeros 300 juegos leídos, conteniendo cada archivo la unión de los 500 comentarios seleccionados al azar del juego en cuestión. Procediendo como antes de esta forma, logramos una forma de obtener la similitud entre diferentes juegos, siendo una posible aplicación el saber cómo de semejante es un juego a uno dado. En nuestro caso, creamos la función *sim_fun_games*, que dada una base de datos construida con una función de escritura a partir de un objeto *bgg_games*, nos proporciona la similitud entre

dos juegos, siendo por ejemplo la semejanza entre el *Colonos de Catán* y el *Dragonmaster* de 0.07.

Podemos aprovechar esta información para obtener una tabla que nos indique los juegos más semejantes a uno dado:

game <chr>	sim <dbl>
Apples to Apples	0.1552550
Downtown	0.1513085
Gateway to the Stars	0.1507817
Can't Stop	0.1495097
Medici	0.1428561
Bohnanza	0.1368918

Figura 5.14: Juegos más semejantes al *Colonos de Catán* con la función *sim_fun_games*.

Fuente: Elaboración propia

Y esos son los juegos más semejantes al *Colonos de Catán* de entre los 300 con los que hemos trabajado. Un análisis rápido nos revela que la similitud en este caso varía entre 0.04 y 0.16, valores razonables teniendo en cuenta que estamos comparando juegos que no tienen por qué compartir siquiera la temática. Si investigamos un poco, vemos que el primer juego *Apples to Apples*, parece ser del mismo estilo familiar que el *Colonos de Catán*, mientras que el segundo, *Downtown*, es un juego de recreación histórica. Este último problema podría resolverse, probablemente, incrementando el número de comentarios y de juegos en el corpus considerado.

5.2 Análisis del sentimiento

Recordemos, antes de nada, que seguiremos los pasos planteados en el capítulo 2 del libro de Kuhn y Silge sobre minería de texto en R [18], en los que exponen algunas formas de abordar este problema.

Para proceder, primeramente recurrimos al modelo de "*tokenización*" más básico considerado en `tidytext`, aquel en el que consideramos que todo el texto no es más que la unión de las diferentes palabras individuales (*tokens*) que lo forman. De esta forma, si bien obtenemos potencia estadística en lo referido a, por ejemplo, el análisis de frecuencias, la perdemos a la hora de entender el significado real del texto, pues no nos es posible distinguir la frase "*very good*" de "*not very good*". Esto puede ser un gran problema para documentos con mucho sarcasmo o un uso constante de la negación,

pero para nuestro trabajo supondremos que es una situación despreciable, lo que se puede corroborar viendo algunos de los comentarios.

Si bien esta no es la forma más eficaz de abordar el problema del procesamiento del lenguaje natural, tal y como acabamos de ver, los paquetes con los que trabajamos están optimizados para llevar a cabo estas tareas, pudiéndose ver esto en los ejemplos de los mismos, y es por ello que, antes de pasar a otros puntos, intentaremos recurrir a estas herramientas. Este proceder es el que recibe el nombre de método basado en diccionarios, y nos permite obtener una puntuación de sentimiento para cada documento, combinando la información obtenida a través de los diferentes unigramas que lo componen, esto es, de las palabras que los forman.

Como mencionamos antes, el paquete `tidytext` dispone de 3 diccionarios basados en unigramas. Cada uno almacena la puntuación asociada a cada palabra de diferentes formas, y es por ello que, antes de nada, resulta conveniente hacer un rápido análisis de cada uno:

- *nrc*, de Saif Mohammad and Peter Turney [8], diccionario que clasifica cada palabra en una o más de las siguientes categorías: *positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise,* y *trust*. Admite Non-Commercial Research Use, aparentemente.
- *bing*, de Bing Liu y colaboradores [5], que es más sencillo que el anterior, y categoriza cada palabra en una de las dos categorías: *positive* o *negative*.
- *afinn*, de Finn Årup Nielsen [9]. De forma semejante al diccionario bing, el *afinn* asigna a cada palabra un único valor, si bien en este caso es numérico y no categórico. Las puntuaciones varían entre -5 y 5, asignándole puntuaciones negativas a sentimientos negativos y procediendo de forma análoga con los positivos.

Volviendo ahora al procesamiento del lenguaje natural, empezamos dándole a los datos, generados a partir de un objeto *bgg_games*, un formato más apropiado para el paquete `tidytext`, siendo el resultado un objeto con el siguiente aspecto, recurriendo a la orden *unnest_tokens*:

index	name	word
1	Die Macher	i
1	Die Macher	like
1	Die Macher	it
1	Die Macher	it's
1	Die Macher	a
1	Die Macher	solid

Figura 5.15: Extracto del resultado de *unnest_tokens*, aplicado sobre un objeto *bgg_games*.

Fuente: Elaboración propia

En el caso expuesto en la Figura 5.15, *name* es el nombre del juego, y junto con *index* identificada inequívocamente al comentario. De esta forma, la descomposición presente en dicha figura está asociada al primer comentario del primer juego, *Die Macher*.

Intentamos ahora combinar estos datos con los que nos proporcionan los diccionarios ya mencionados. Recurriendo, a modo de ejemplo, al sentimiento alegría del diccionario *nrc*:

```
nrc_joy <- get_sentiments("nrc") %>%
  filter(sentiment == "joy")

bgg_games_comments %>%
  inner_join(nrc_joy, by = "word") %>%
```

index <int>	name <chr>	word <chr>	sentiment <chr>
6	Die Macher	gift	joy
7	Die Macher	luck	joy
9	Die Macher	fun	joy
11	Die Macher	good	joy
15	Die Macher	good	joy
15	Die Macher	score	joy

Figura 5.16: Selección del sentimiento alegría del diccionario *nrc* con *get_sentiment* e *inner_join* para generar la tabla.

Fuente: Elaboración propia

Obtenemos un *tibble* que compacta toda la información que necesitamos, si bien habremos de generalizar el procedimiento, aplicándolo a todos los sentimientos

disponibles en el diccionario *bing*, seleccionado al proporcionarnos varios sentimientos, además del ser positivo o negativo, información que habremos de usar en los siguientes pasos que demos.

Con un *inner_join,* como hicimos antes, obtenemos una tabla con el siguiente aspecto, calculando *sentiment* como la diferencia entre *positive* y *negative*:

name <chr>	index <int>	negative <int>	positive <int>	sentiment <int>
Die Macher	1	2	0	-2
Die Macher	5	0	1	1
Die Macher	6	1	2	1
Die Macher	7	0	1	1
Die Macher	9	0	1	1
Die Macher	11	0	1	1

Figura 5.17: Generación de sentimiento para cada comentario.

Fuente: Elaboración propia

Notemos que el no aparecer ninguna palabra del diccionario en un comentario hace que dicho comentario tenga sentimiento asociado 0, desapareciendo de nuestra tabla al haber recurrido al *inner_join* como paso intermedio.

Visualizamos ahora, para algunos juegos, los resultados obtenidos siguiendo el proceder aquí expuesto:

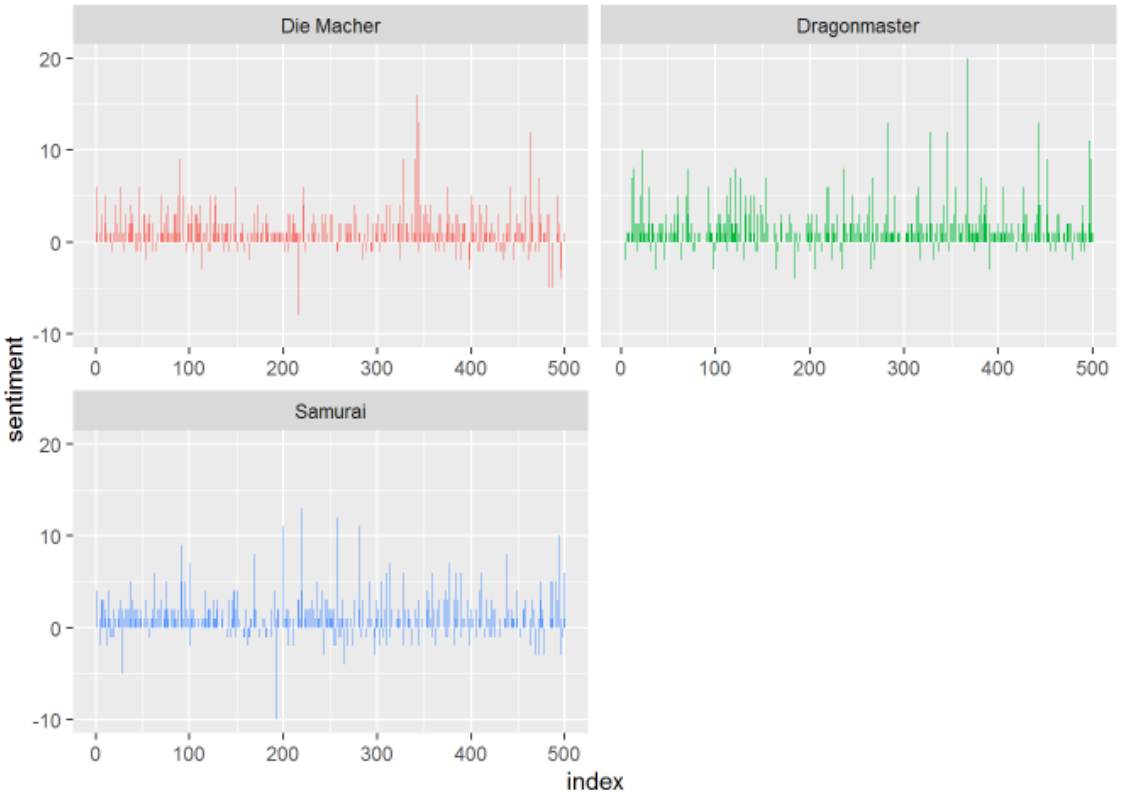

Figura 5.18: Representación del sentimiento para 3 juegos, por comentarios.

Fuente: Elaboración propia

En general, observamos que el sentimiento es mayoritariamente positivo para todos los juegos, dándose algunos extremos, principalmente por ser positivos, pero también en algunos casos por ser especialmente negativos, como ser puede ver en el caso del juego *Samurai*, donde el comentario 200, aproximadamente, es detectado como muy negativo con este proceder. Con esta información podemos ver la percepción general del público, e identificar también anomalías como la del comentario mencionado. Más adelante, en el capítulo 6, incorporaremos estos resultados como variable auxiliar, que resumirá a la variable *comment*, con la que no es posible trabajar directamente.

Con los *inner_join* hemos podido perder, como decíamos, comentarios enteros, pero como en esos casos el sentimiento asociado es 0, los gráficos anteriores siguen siendo válidos.

Antes de continuar, podemos ojear las matrices de frecuencias de un juego y de varios, para comprobar la estructura de los datos con los que vamos a trabajar, teniendo en cuenta que a la hora de construirlos siempre vamos a recurrir a un *inner_join*. Por ejemplo, para los tres juegos considerados:

name <chr>	word <chr>	n <int>
Dragonmaster	fun	80
Die Macher	fun	75
Samurai	fun	68
Samurai	good	61
Die Macher	good	53
Dragonmaster	good	49

Figura 5.19: Extracto de la matriz de frecuencias para las palabras de nuestro vocabulario en los 3 juegos selecionados.

Fuente: Elaboración propia

Ahora, con lo obtenido podemos generar un valor de sentimiento promedio para cada juego, que luego incluiremos en la modelización como alternativa a los comentarios, como ya mencionábamos:

name <chr>	sentiment_mean <dbl>
Samurai	1.621302
Dragonmaster	1.390909
Die Macher	1.285714

Figura 5.20: Sentimiento medio asociado a los juegos estudiados.

Fuente: Elaboración propia

Como pudimos ver en la Figura 5.18, el sentimiento puede variar entre -20 y 20, si bien casi siempre se encontrará en la franja entre -3 y 5. Teniendo esto en cuenta, es esperable encontrarnos con medias del sentimiento positivas, al menos para los casos considerados, indicando de esta forma que el público ha recibido bien el producto en cuestión.

En general, un sentimiento promedio elevado indicará que el juego está recibiendo buenas críticas, aunque dicho dato luego no se vea presente en otras variables que deberían determinar el valor de un juego, pues puede darse, por ejemplo, que sea un gran juego pero que sea infrecuente o caro, en cuyo caso sería esperable encontrar, igualmente, un gran número de comentarios positivos.

Estos resultados, expresados de esta forma gráfica, nos permiten interpretar, con rapidez, cuál es el sentimiento generalizado respecto de un juego particular. Una forma de aplicar esto, además de para condensar la información relativa a los juegos, es

recurrir a la creación de una nueva variable que guarde el sentimiento producido por un juego, para cada juego incluido en cierto modelo. Podemos, además, para ver lo acertado que podría ser nuestro análisis, comparar el sentimiento de cada juego con el rating que recibió, para el juego *Die Macher*, por ejemplo.

5.2.1 Aplicación: *sentiment* como predictor de la puntuación en *Die Macher*

Una vez hemos generado el *tibble* en el que aparecen *sentiment* y *rating* correctamente asociados, primeramente hemos de normalizar para poder comparar correctamente los resultados, y a continuación definimos la diferencia para aplicar contrastes de normalidad para dos muestras relacionadas. Representamos el histograma para ver qué aspecto tiene y hacernos una idea sobre la normalidad:

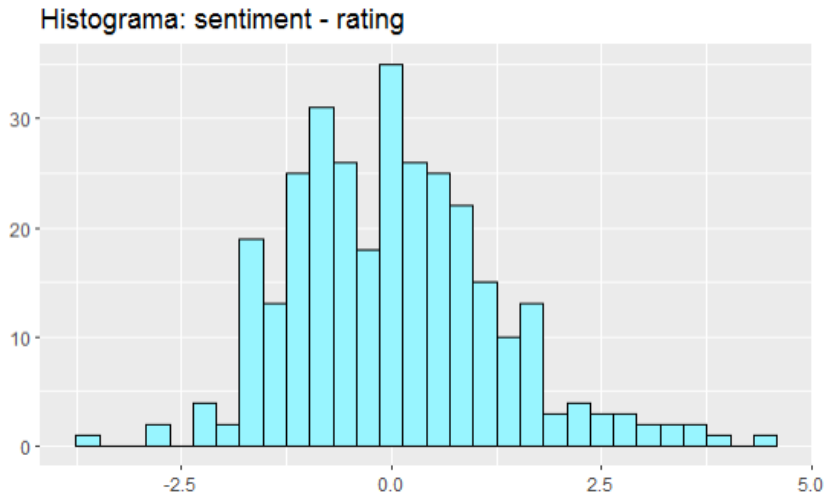

Figura 5.21: Histograma de la diferencia entre el sentimiento generado y *rating*.

Fuente: Elaboración propia

Podríamos aceptar un comportamiento más o menos normal, a pesar de la bajada que sufre el histograma en valores próximos a 0, dándole un aspecto más bien bimodal.

Aplicando ahora el Test de Shapiro-Wilk obtenemos un p-valor prácticamente nulo, $3.73 \cdot 10^{-5}$, luego rechazamos la hipótesis nula, de normalidad, no pudiendo por lo tanto suponer la normalidad de la diferencia, a pesar de la intuición que nos podía transmitir el histograma.

Acudiendo ahora a los contrastes no paramétricos, el Test de Wilcoxon nos proporciona un p-valor de 0.701, luego no podemos rechazar ni la simetría ni que la mediana se encuentre en el 0, luego según esta prueba no existe diferencia apreciable entre el rating obtenido y el esperado basándonos únicamente en el análisis de sentimiento. Este

resultado resulta increíble, pues afirma que nuestro sentimiento ha servido como representación del contenido y de la intención del comentario.

Una alternativa sería plantear el Test ANOVA directamente sobre el conjunto de datos generado, en cuyo caso obtenemos un p-valor prácticamente nulo, $1.2 \cdot 10^{-6}$, afirmando que el sentimiento influye en el rating, lo cual valida hasta cierto punto nuestro modelo de análisis del sentimiento. Concretamente, viendo el resumen del modelo lineal nos encontramos con que cada punto de sentimiento incrementa en 0.14 el rating estimado, si bien obtenemos un R^2 de 0.03, luego como predictor individual no es demasiado bueno, a pesar de ser capaz de explicar la variabilidad.

Intentamos, sin embargo, combinar esta información con la de otro juego, para ver si con más variables podemos obtener un mejor resultado, guardando este valor en un documento RData aparte, para aplicarlo en el modelo general en el capítulo 6.

5.2.2 Comparación de diferentes diccionarios

Comparemos ahora los diferentes diccionarios a los que podríamos haber recurrido, para ver si los resultados son muy diferentes. Centrándonos en el primer juego, *Die Macher*:

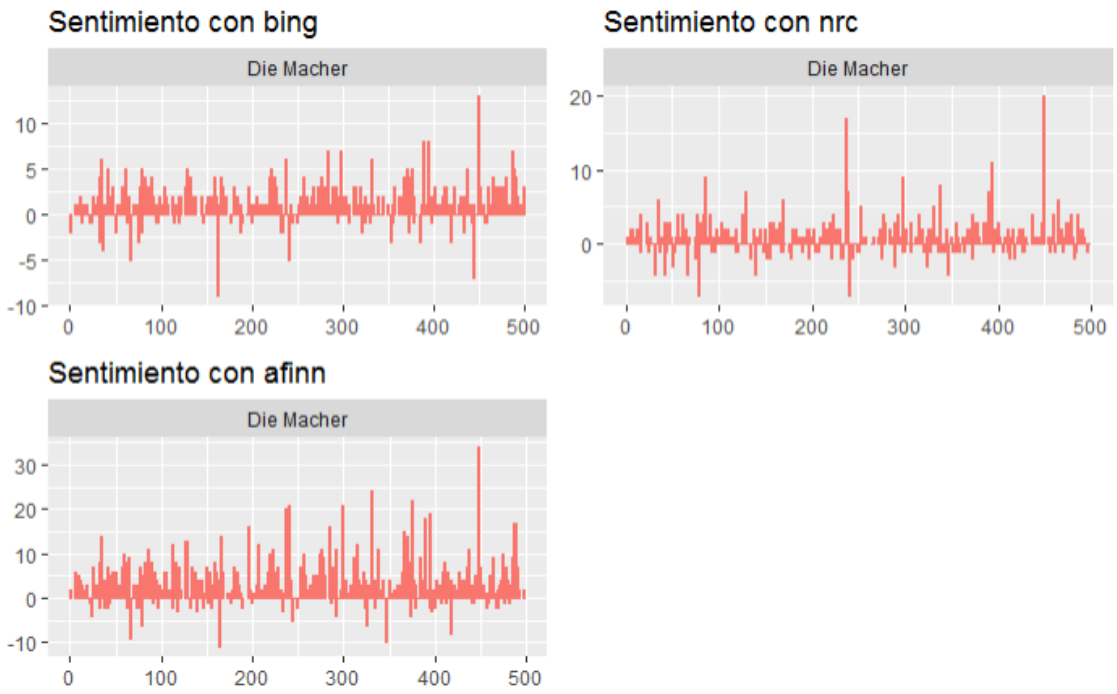

Figura 5.22: Sentimiento dependiendo del diccionario considerado para *Die Macher*.

Fuente: Elaboración propia

Obtenemos resultados muy semejantes en forma para los tres, a pesar de que con *afinn* y *nrc* los valores son mayores que con *bing*, luego al final nuestra decisión no es tan relevante como parecía al principio, siempre y cuando estemos trabajando con valores normalizaados. De esta forma, optamos por recurrir a un conjunto o a otro dependiendo de nuestros intereses, si bien recurriremos por lo general a *nrc* al haberse diseñado para trabajar con datos de Twitter, pues consideramos dicha plataforma como próxima a la BGG. De hecho, recurriendo a [33] podemos ver que hay un gran número de internautas que "rebotan" de la BGG a Twitter, aproximadamente del 10%.

5.2.3 Más allá: sentimientos personalizados

Otra forma de orientar el estudio, aprovechando la información contenida en el diccionario *nrc*, sería explotar “sub-sentimientos”, esto es, agrupaciones de sentimientos que se corresponden con la respuesta natural a una situación dada. Así, por ejemplo, sería útil saber si un juego es adecuado para una persona que sufra miedo con facilidad, basándose únicamente en un análisis gráfico de las palabras más comunes cargadas de sentimiento, agrupadas por categoría. Este será justamente nuestro objetivo en esta sección, la creación e interpretación de sentimientos personalizados, que agrupen a otros y que se ajusten a una solicitud concreta.

Recordemos ahora que los sentimientos que nos proporciona *nrc* son: confianza, miedo, negativo, tristeza, furia, sorpresa, positivo, disgusto, alegría y anticipación. Con cada una de esas categorías podríamos construir un sentimiento artificial, como el ejemplo expuesto antes. De esta forma, si buscamos un juego que genere alegría y confianza pero que no incluya ni sorpresas ni miedo, siguiendo el ejemplo del miedo antes mencionado, podemos construir:

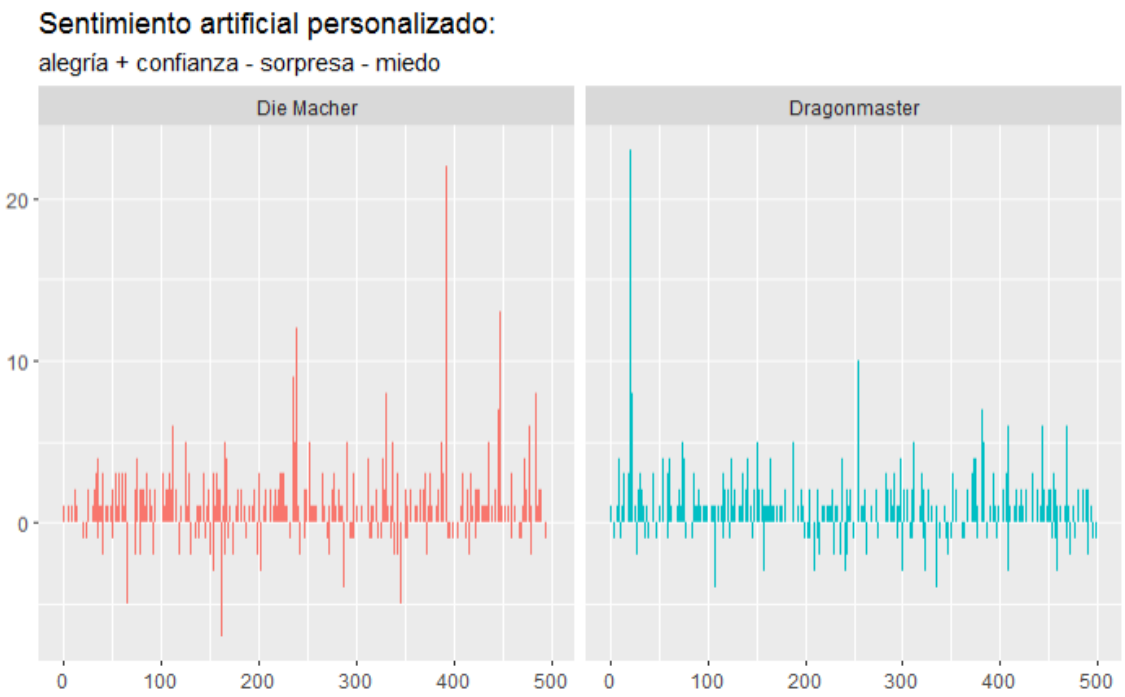

Figura 5.23: Sentimiento personalizado para dos juegos.

Fuente: Elaboración propia

Así, podemos ver que este tipo de miedo puede estar más presente en el segundo juego, dándose un mayor número de picos positivos en el primero, indicando que de encontrarnos en la disyuntiva de escoger uno u otro, *Die Macher* sería la elección, si bien deberíamos mirar los comentarios más extremos para obtener información adicional.

Una limitación de este proceder, sin embargo, es que el vocabulario propio del juego puede condicionar en gran medida el sentimiento generado, de forma tal que es probable que en *Dragonmaster* aparezcan palabras de combate o dragones, asociadas con toda seguridad en *nrc* al miedo o a la sorpresa. Una mejor forma de solventar este problema, al menos en parte, es el tener cuidado a la hora de seleccionar el vocabulario, teniendo en cuenta la categoría a la que pertenece, por ejemplo.

Otra forma de representar esta información mucho más gráfica y visualmente atractiva, es la que nos proporciona la orden *comparison.cloud* del paquete `wordcloud`:

Figura 5.24: Nube de palabras por sentimientos para el juego *Samurai*.

Fuente: Elaboración propia

De esta forma no solo podemos saber qué sentimientos podría generar cierto juego, sino que además conocemos más calificativos para cada uno, pudiendo saber con precisión de qué tipo de juego se trata. Vemos, en este caso concreto, la importancia del diccionario, pues palabras como "*dungeon"* y "*opponent"* aparecen catalogadas eb miedo, así como *system* en confianza, si bien es probable que la intención de estas palabras sea diferente a la esperada por el diccionario.

Igualmente, con esto hemos logrado una representación que compacta con rapidez los comentarios de un juego, proporcionándonos no solo las palabras más comunes sino también agrupadas por categoría, lo que junto con la creación de sentimiento artificial podrían ser una gran ayuda a la hora de obtener información sobre un juego seleccionado.

5.2.4 Análisis de frecuencias: aportación al sentimiento

Notemos que hemos dejado para esta sección el análisis de frecuencias como tal, pues es evidente que ejecutando algo como lo anterior podemos obtener grandes resultados no numéricos, que le den al lector una idea general de los comentarios que suele recibir el juego, pudiendo luego inferir este sobre las características del mismo.

Si queremos ver, por ejemplo, las palabras más frecuentes agrupadas por sentimiento podemos recurrir a una gráfica como la Figura 5.25:

word <chr>	sentiment <chr>	n <int>
fun	positive	223
like	positive	196
good	positive	163
great	positive	122
well	positive	78
better	positive	74

Figura 5.25: Palabras más comunes y sentimiento asociado en el juego *Samurai*.

Fuente: Elaboración propia

Basándonos en esta última figura, y conectando con la sección anterior, podríamos emplear tablas como la de la Figura 5.25, para cada sentimiento, para comprobar que la asignación de sentimientos es correcta dada la naturaleza del juego seleccionado, si bien es cierto que las palabras presentes, las positivas más comunes en varios juegos, en rara ocasión tendrán una connotación diferente. Para contrastar esto, y para seguir aprovechando la información generada, podemos ver cuáles han sido las palabras que más han aportado al sentimiento total al aparecer más veces:

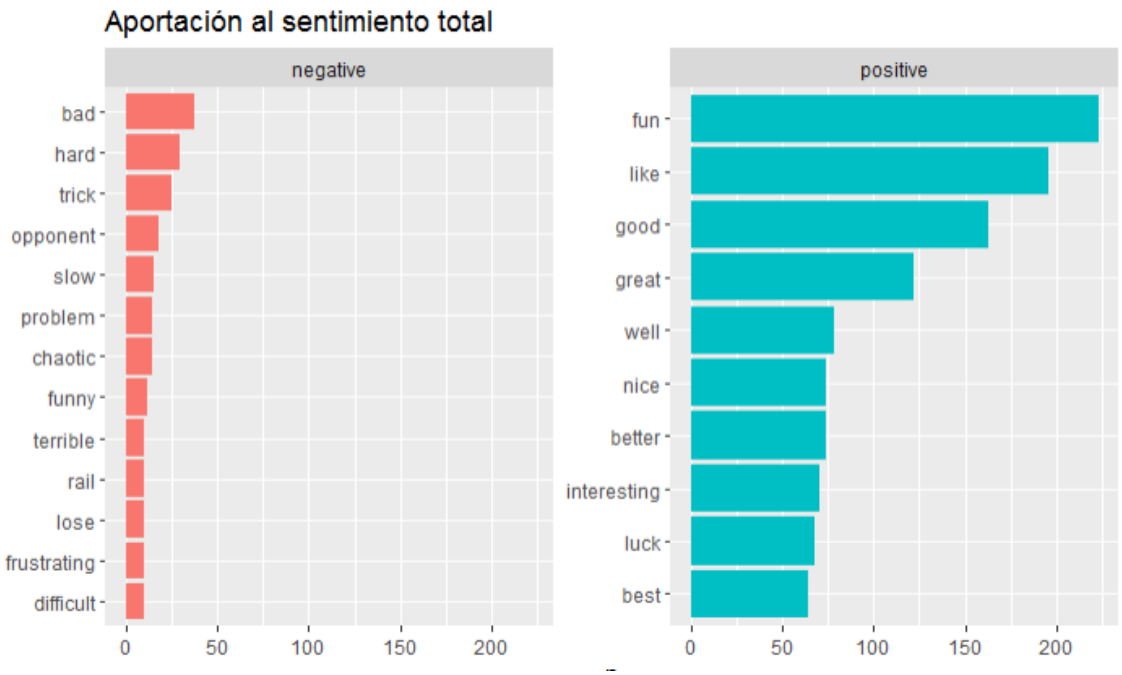

Figura 5.26: Aportación al sentimiento total en el juego *Samurai*.

Fuente: Elaboración propia

Así, podemos ver primeramente que, tal y como esperábamos, hay una mayor frecuencia de palabras positivas, pues recordemos que el sentimiento promedio de este juego era próximo a 2. Por otro lado, aunque pocas palabras nos sorprendan en el campo *positive*, en *negative* varias podrían ser suprimidas, como *opponent* o *lose*, si bien para tomar esta decisión sería necesario un análisis más en profundidad de los comentarios seleccionados, quizás tomando una muestra mayor y estudiando tanto las categorías como las mecánicas del juego.

5.2.5 Otras funcionalidades con `wordcloud`

Aprovechamos este momento para suprimir otras palabras que deberíamos eliminar, como *play*, *players* y *game*, entre otras que no aportan información alguna en nuestro análisis del sentimiento, y que de hecho pueden corromperlo. Hecho esto obtenemos la siguiente nube de palabras para el mismo juego:

Figura 5.27: Nube de palabras para *Samurai*, tras un filtro de palabras.

Fuente: Elaboración propia

Como ya mencionábamos, esta gráfica resume de forma rápida y eficiente el posible contenido de un juego, si bien podemos ir más allá, ejecutando algo como lo que hicimos para los sentimientos artificiales, y obteniendo de esta forma la siguiente figura:

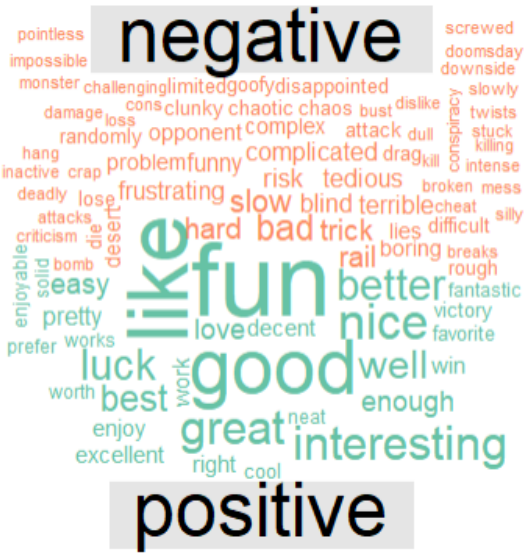

Figura 5.28: Nube de palabras por tipo de sentimiento para el juego *Die Macher*.

Fuente: Elaboración propia

Esto nos permite ver que el juego es, por ejemplo, lento, si observamos las palabras negativas más frecuentes. Esta información desglosa aún más el contenido de la anterior nube de palabras, y podría ser un gran añadido a las funcionalidades de la BGG, o de cualquier otra plataforma que desee proporcionar información sobre un juego concreto.

5.2.6 Nombres

Podemos repetir algo como lo que hemos hecho hasta ahora para los nombres de los juegos y para las descripciones, descubriendo en el proceso un resultado más que llamativo, como veremos en la siguiente sección. En estos casos, como trabajamos únicamente con un campo de cada entrada de los datos, podemos trabajar con la totalidad de los datos descargados, con una pequeña adaptación de la función *read_bgg.*

Si hacemos como antes y vemos la siguiente nube de palabras, para los nombres de los juegos, obtenemos un primer resultado interesante, y es que hay una clara tendencia dentro del mercado de los juegos de mesa a lo bélico y fantasioso, con palabras como aventura, horror, guerra o batalla destacando frente a otras.

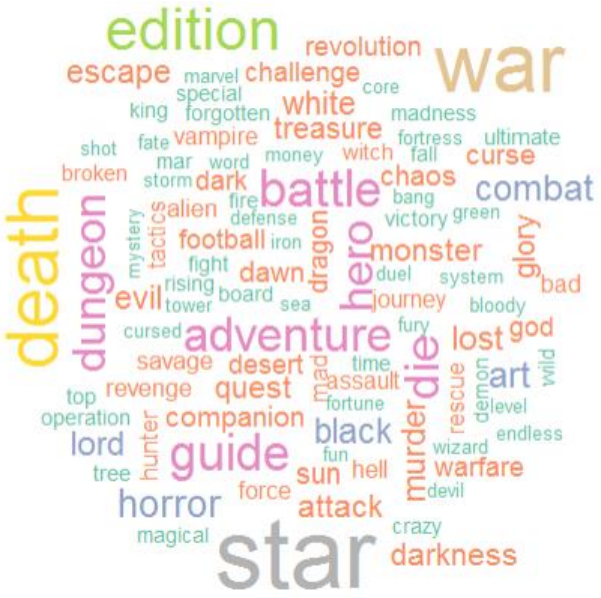

Figura 5.29: Nube de palabras para los nombres de todos los juegos.

Fuente: Elaboración propia

Podemos ir más allá, generando de forma análoga a la anterior el sentimiento asociado a un juego, obteniendo de esta forma una tabla como la que sigue:

sentiment <int>	average <dbl>	name <chr>
1	7.27273	'65: Action Card Expansion
0	7.43333	'65: Alone in the Jungle
0	7.67966	'65: Squad-Level Combat in the Jungles of Vietnam
1	8.00000	'Allo 'Allo! Cartoon Fun
-1	5.66667	'Attack of the Boogeymen' Supplement (fan expansion for Last Night on Earth)
0	5.27523	'CA' Tactical Naval Warfare in the Pacific, 1941-45

Figura 5.30: Sentimientos y puntuaciones para diversos juegos.

Fuente: Elaboración propia

Ahora, intentando copiar lo hecho previamente, vemos si es viable construir un modelo lineal que, dado el sentimiento de un juego nos proporcione su puntuación. Aunque sepamos que este resultado sería impensable, resulta igualmente interesante comprobarlo, aprovechando que tenemos los datos preparados. Por desgracia, obtenemos con el modelo lineal un R^2 del orden de 10^{-5}, indicándonos que, tal y como esperábamos, el nombre no explica por sí solo la puntuación de un juego.

Sin embargo, sí es cierto que podría existir cierto comportamiento general dentro de los nombres de los juegos, para cierto subgrupo, y eso es justamente lo que estudiaremos en el siguiente punto.

5.2.7 Aplicación: ¿ser *lovecraftiano* influye?

Aunque este punto se corresponda con un análisis de la varianza, siendo lo correcto incluirlo en la sección 6.3, optamos por dejarlo aquí al tener una mayor relación con la sección que acabamos de ver, exponiendo resultados más allá de los vistos en la última sección.

Yendo más allá de lo que hicimos antes, intentemos ver si un nombre de juego que sea "*lovecraftiano*" condiciona directamente la puntuación, definiendo *lovecraftiano* como que incluye las palabras "Cthulhu", "Lovecraft", "*Old Ones*" y variaciones de dichas palabras en mayúsculas y en minúsculas.

Obtenemos con el Test ANOVA un p-valor prácticamente nulo, $3.8 \cdot 10^{-6}$, luego podemos descartar la hipótesis nula de que no influye, si el p-valor es por ejemplo menor a 0.05. Comparando ahora las medias de forma descriptiva:

lovecraftian <lgl>	average <dbl>
FALSE	6.638427
TRUE	7.109770

Figura 5.31: Puntuación y sentimiento por categorías.

Fuente: Elaboración propia

Aprovechamos para aclarar que no hicimos el mismo contraste para el sentimiento promedio al poder ser el resultado aberrante, dado que el propio vocabulario *lovecraftiano* tiende a ser tétrico, lo que condicionaría considerablemente el resultado obtenido, desvirtuándolo.

Volviendo a la comparación de la puntuación por categorías, recurriendo a un Test T obtenemos un p-valor del orden de 10^{-6}, luego rechazamos la hipótesis nula de no efecto de la variable *lovecraftian*, esto es, misma media en las categorías, lo que nos permite afirmar que el ser o no *lovecraftiano* influye en el resultado. Concretamente, el Test T nos proporciona un intervalo de confianza entre 0.30 y 0.64, indicándonos de

esta forma que el mero hecho de tener nombre *lovecraftiano* ya puede proporcionarle al juego entre 0.3 y 0.6 puntos en la valoración media.

5.2.8 Descripciones

Por completar el análisis de los textos considerados, repetimos lo hecho para los nombres pero para las descripciones de los juegos:

Figura 5.32: Nube de palabras para las descripciones de todos los juegos.

Fuente: Elaboración propia

Comprobamos una propiedad interesante, y es que parece que se emplea el mismo vocabulario en los nombres y en las descripciones, con la salvedad de la palabra *player*, claro, y es que en este caso también hemos obtenido como palabras con mayor peso las asociadas a la aventura, el combate, los héroes y el poder.

Una forma de aprovechar este resultado sería construyendo un nuevo valor lógico para cada juego, que nos indicase si su contenido, basándose en la descripción, es semejante al de otros juegos comunes, pudiendo recurrir al modelo vectorial de la sección 5.1 para alcanzar este objetivo.

5.2.9 Más que unigramas: el modelo de ratios

Abordamos en este punto frases como "*I am not having a good day*", expresiones de sentimiento negativo global que al ser procesadas por nuestro modelo de análisis de

sentimiento nos proporcionarían resultados erróneos, pues al tratar únicamente con unigramas nos centramos en palabras como "*good*", sin tener en cuenta posibles negaciones o sarcasmos.

Las ratios que construiremos nos permiten descomponer de una forma ligeramente mejor, si bien es cierto que realmente lo conveniente sería recurrir a otro motor semántico más potente, que nos permitiese diferenciar NLP de la estadística que podemos aplicar. Sin embargo, esto se sale del alcance de este trabajo de fin de grado, en el que se ha optado por realizar una incursión interesante en este campo desconocido hasta ahora para mí, pero no siendo el planteamiento principal sino parte de un conjunto de objetivos más global.

Empezamos descomponiendo los diferentes comentarios en frases en lugar de en palabras individuales con *unnest_tokens*, cambiando el valor del parámetro *token*:

index	name	sentence
1	Die Macher	i like it.
1	Die Macher	it's a solid game, most every time i've played it, which is a pretty decent compliment.
1	Die Macher	i love the double-headed queue and bidding for the shops.

Figura 5.33: Descomposición del primer comentario en frases.

Fuente: Elaboración propia

Aprovechamos para comentar que *unnest_tokens* admite también el uso de patrones, de expresiones regulares, si bien es cierto que para nuestros intereses no será necesario recurrir a semejante capacidad.

Obtenemos ahora las ratios de palabras negativas por frase, tal y como las definimos en la sección 2.5.4, tomando la media dentro de cada comentario:

name	index	negative_words	words	ratio
Die Macher	5	1	2	0.500
Die Macher	10	1	8	0.125
Die Macher	13	1	1	1.000
Die Macher	16	1	4	0.250
Die Macher	17	1	1	1.000
Die Macher	20	1	2	0.500

Figura 5.34: Ratios: palabras negativas por comentario.

Fuente: Elaboración propia

Como hicimos antes, intentamos tomar la ratio como predictor, para el juego *Samurai*. Aplicando el Test ANOVA, obtenemos un p-valor de 0.56, luego no podemos afirmar que la ratio influya en la puntuación del comentario, al menos no individualmente, pudiendo combinarse con otras variables para obtener mejores resultados.

Sin embargo, este valor nos proporciona información como antes el sentimiento, dándonos una idea aproximada de los negativos o positivos que pueden llegar a ser los comentarios de un juego.

6 MODELIZACIÓN

En este apartado nos centramos en el aprovechamiento de los datos tratados a lo largo del proyecto, con el objetivo de construir modelos basados en las diferentes variables que hemos leído, con el objetivo de ser capaces de predecir el valor de la variable *average* dadas las otras de los conjuntos *bgg_games*. Además, construiremos los mismos modelos pero con menos variables, pues hay campos que no resulta plausible obtener a priori, como el número de usuarios que lo tienen o que lo quieren —respectivamente, las variables *wanting* y *whishing* —, si bien incluiremos aquellas que son relativamente fáciles de estimar con una muestra lo suficientemente grande de personas que prueben el juego. Catalogamos dentro de este grupo las variables *averageweight* o *language_dependence*, entre otras.

Junto con cada modelo, como es reglamentario, incluiremos una estimación del error que podríamos llegar a cometer con el mismo, realizándose esta estimación por validación cruzada siempre que nos sea posible.

Notemos que en este caso las dimensiones de los datos son un problema considerable, pues trabajamos con un gran número de variables, muchas de ellas provenientes de estructuras de datos anidadas, y debemos preprocesar varios campos para poder trabajar con según qué modelos. Es por ello que tomaremos los primeros 100 documentos XML que generamos en la sección 3.2, obteniendo de esta forma un subconjunto de 29.925 juegos, con 25 variables para cada entrada, si bien tendremos que descartar algunas entradas al tener campos NA, lo que imposibilita su uso en los modelos. Sin embargo, en aquellos casos en los que el valor que nos falte sea *average*, intentaremos conservar dicha entrada para comparar los resultados con otras entradas semejantes.

6.1 Preprocesamiento de los datos

Como ya mencionábamos en la sección anterior, antes de aplicar diferentes modelos debemos dar varios pasos previos en materia de preparación de los datos. En primer lugar, una vez hemos cargado el conjunto *bgg_*games, con casi 300.000 filas, podemos aprovechar el vector *sentiment_mean* generado en el capítulo anterior, incluyéndolo como una variable predictora más.

```
bgg_games %<>% left_join(bgg_games_comments_sentiment_mean, by = "name")
bgg_games$sentiment_mean[is.na(bgg_games$sentiment_mean)] = 0
bgg_games %<>% select(-c(comment, ranks))
```

Figura 6.1: Añadimos los sentimientos promedios a nuestros datos, como variable.

Fuente: Elaboración propia

Dado que en el capítulo anterior trabajamos únicamente con una muestra de los juegos, y en este apartado trabajaremos con una mayor, nos vemos obligados o bien a trabajar con un gran número de entradas con NA en *sentiment_mean*, o bien a asignar un valor por defecto para todos los campos vacíos. Optaremos por la segunda opción, pues resulta aceptable considerar que todo juego tiene un sentimiento asociado neutro hasta que se demuestre lo contrario, siendo esto por ejemplo verdad para juegos con pocos o ningún comentario, y pudiendo asumirse al haber obtenido a lo largo del trabajo valores de *average_mean* que varían entre 1 y 3.

Una vez hemos hecho esto, el siguiente paso es trabajar con las familias, categorías y mecánicas, pues recordemos que ya vimos en la Figura 5.34 que había más de 1.000 familias, 160 mecánicas y 80 categorías, siendo por tanto imperativo el trabajar con una subselección, de forma tal que a la hora de construir las variables *dummy* no acabemos creando un *tibble* con 2.000 columnas, pues esto puede llegar a ser un problema para según que modelos. Aprovechamos para recordar un detalle, y es que cada juego puede tener cero, una o más de cada una de dichas categorías, luego la creación manual de variables *dummy* es imperativa, siendo inviable el recurrir iterativamente a *unnest_longer* del paquete `tidyr`, de `tidyverse`, pues en ese caso tendríamos, para cada entrada, hasta 100 o más entradas asociadas en la versión desanidada. Concretamente, que cada entrada pueda pertenecer a más de una categoría hace imposible el llevar a cabo un proceso automático de *one-hot-encoding*, obligándonos a preparar estas nuevas variables antes de incluir los datos en el *workflow* del modelo, asignando 1 en cada categoría a la que se pertenezca y 0 a las otras.

Por lo tanto, hacemos una selección de familias y de mecánicas, reformulando aquellas vacías como *"Without family/category/mechanic"*, tal y como podemos ver en el siguiente extracto de código:

```
common_families = bgg_games %>%
    unnest_longer(boardgamefamily) %>%
    select(boardgamefamily, name)  %>%
    filter(boardgamefamily != "Admin: Better Description Needed!") %>%
    mutate(boardgamefamily = ifelse(is.na(boardgamefamily),
                                    "Without family",
                                    boardgamefamily)) %>%
    group_by(boardgamefamily) %>%
    count %>%
    ungroup() %>%
    slice_max(n = n_max_families, order_by = n) %>%
    select(boardgamefamily) %$%
    boardgamefamily
```

Figura 6.2: Selección de las *n_max_families* familias más comunes.

Fuente: Elaboración propia.

Notemos, adicionalmente, que hemos descartado la variable *ranks,* así como la familia de la que ya habíamos hablado antes, aquella que no era más que una nota de los administradores. Sobre ranks, mencionamos que es una variable que se deriva de la combinación de puntuación y de familia o categoría, y que también se encuentra anidada, luego optamos por descartarla, dándole preferencia a otras. Nuestra idea con el objeto *common_families* es poder recodificar las tres variables mencionadas, asignando la categoría "*Other"* para los juegos que no pertenezcan a estas familias, lo que facilitará considerablemente la interpretación de los modelos.

Tomamos, para cada una de estas categorías, los 10 niveles más comunes, construyendo de esta forma los vectores *common_families, common_mechanics* y *common_categories*, siendo estos dos últimos análogos en definición a *common_families.*

Modificamos ahora el *tibble bgg_games* para que se adecúe a nuestras necesidades, modificando multitud de variables para mejorar la interpretación, y otras, como las ya mencionadas, para obtener modelos válidos y con los que R pueda trabajar. Incluimos en las siguientes figuras algunos ejemplos del código empleado para realizar esta preparación de los datos:

```
bgg_games %<>%
  mutate(aux = TRUE) %>%
  unnest_longer(boardgamecategory) %>%
  mutate(boardgamecategory = ifelse(is.na(boardgamecategory),
                                    "None",
                                    boardgamecategory)) %>%
  mutate(boardgamecategory = ifelse(boardgamecategory %in% common_categories,
                                    boardgamecategory,
                                    "Other category")) %>%
  pivot_wider(names_from = boardgamecategory,
              values_from = aux, names_prefix = "category:",
              values_fn = function(x) {!is.null(x)},
              values_fill = FALSE) %>%
```

Figura 6.3: Preparación de las categorías, recodificando la ausencia y aquellas que no estén en el vector *common_categories.* Con *pivot_wider* construimos las variables dummy.

Fuente: Elaboración propia.

Para *suggested_playerage* y *language_dependence* el procedimiento es semejante, pero más sencillo:

```
mutate(aux = TRUE) %>%
mutate(suggested_playerage = ifelse(is.na(suggested_playerage),
                                    "Unknown",
                                    suggested_playerage)) %>%
pivot_wider(names_from = suggested_playerage,
            values_from = aux,
            names_prefix = "suggested_playerage:",
            values_fill = FALSE) %>%
```

Figura 6.4: Preparación de las *suggested_playerage*, recodificando la ausencia y construyendo manualmente las variables *dummy*.

Fuente: Elaboración propia.

Siguiendo con el preprocesamiento, se nos presenta la disyuntiva de normalizar o no las variables numéricas incluidas en los datos para mejorar el comportamiento de los modelos y asegurar que no asignan pesos basándose en órdenes de magnitud. Aunque nos gustaría modificar las variables sin necesariamente normalizarlas, obteniendo de esta forma resultados de una interpretación más sencilla, tras comprobar el comportamiento de los diferentes modelos que consideraremos ante los dos casos expuestos, nos encontramos con que el normalizar mejora considerablemente el rendimiento de los modelos incluidos en las siguientes secciones, y es por ello que finalmente decidimos normalizar, ejecutándose este paso en la receta de la Figura 6.7.

Por último, descartamos el campo *name* al ser simplemente un identificador, así como las filas que contengan NA con la orden *drop_na*, y de esta forma ya tenemos un

conjunto de datos preparado para aplicar los diferentes modelos de la siguiente sección. Para evitar problemas de colinealidad, descartamos las columnas *suggested_playerage:21+*, *language_dependence: Unplayable in another language*, pues están asociadas a categorías mutuamente excluyentes y podrían estar generando problemas en los ajustes. Con los pasos indicados en las últimas líneas, el conjunto generado presenta el siguiente aspecto, para una conveniente selección de columnas:

average <dbl>	category:Card Game <lgl>	family:Players: Two Player Only Games <lgl>	mechanic:Dice Rolling <lgl>
7.62256	FALSE	FALSE	TRUE
6.64398	TRUE	FALSE	FALSE
7.45017	FALSE	FALSE	FALSE
6.59769	FALSE	FALSE	FALSE
7.33818	FALSE	FALSE	FALSE
6.51667	FALSE	FALSE	TRUE

Figura 6.5: Aspecto de *bgg_games* tras el preprocesado manual.

Fuente: Elaboración propia

Buscamos, como ya mencionamos, seguir la metodología presentada por el paquete `tidymodels`, razón por la cual ahora construiremos los conjuntos para llevar a cabo la validación cruzada, y justo después construiremos una receta, en el sentido del paquete *recipe*, para incluirla dentro del flujo de trabajo (*workflow*) general que crearemos, adaptándolo, en su caso, para cada modelo si fuese necesario.

```
bgg_games_split = initial_split(bgg_games, prop = 0.8)
train_data = training(bgg_games_split)
test_data = testing(bgg_games_split)
cv_folds = vfold_cv(train_data, v = 10, breaks = 4)
```

Figura 6.6: Separación en conjunto de entrenamiento, test y pliegues para la validación cruzada con funciones del paquete *rsample,* de `tidymodels`.

Fuente: Elaboración propia

6.1.1 Preparación de la receta

Vemos a continuación la receta que ya habíamos mencionado, en la que primeramente recurriremos a un filtro de varianza, *step_zv,* que elimina aquellos campos que sean constantes, para después normalizarlos con *step_center* y *step_scale*, finalizando la preparación con la omisión de posibles NA que no hayamos tratado, tanto en el conjunto de entrenamiento como en el conjunto test, y suprimiendo aquellas entradas en las que

average valga 0, pues esa es la forma en la que la BGG codifica que nadie ha realizado aún valoración alguna de dicho juego.

```
my_rec <- recipe(average ~ ., data = train_data) %>%
  step_zv(all_predictors()) %>%
  step_naomit(all_predictors(), skip = FALSE) %>%
  step_center(all_numeric(), -all_outcomes()) %>%
  step_scale(all_numeric(),  -all_outcomes()) %>%
  step_filter(average != 0)
```

Figura 6.7: Receta general con la que trabajaremos.

Fuente: Elaboración propia

Podríamos declarar adicionalmente interacciones entre variables, como la que podría existir entre el número mínimo y el máximo de jugadores, o diferentes categorías y mecánicas, donde sí que se podría dar esta relación de dependencia entre variables. Sin embargo, varios intentos nos revelaron que haciendo estos cambios solo lográbamos sobreajustar los datos, incrementándose todas las métricas de error. Es por ello que planteamos el modelo sin interacciones.

Finalmente, para declarar nuestro flujo de trabajo e incluir esta receta dentro del mismo, simplemente recurrimos a la siguiente orden:

```
my_wf = workflow() %>% add_recipe(my_rec)
```

Figura 6.8: Declaración del flujo de trabajo general.

Fuente: Elaboración propia

De esta forma, ya tenemos preparados los datos, la receta y el flujo de trabajo, siendo necesarias en todo caso pequeñas modificaciones para cada uno de los casos que consideraremos en las siguientes secciones.

6.2 Modelo general

Incluimos en esta sección la construcción de modelos de predicción clásicos, trabajando con todas las variables o con subconjuntos seleccionados, con el objetivo de predecir *average.* Usaremos, siempre que sea posible, el flujo de trabajo definido en la sección previa, modificado en caso de ser necesario con la orden *update_recipe*.

6.2.1 Sobre las métricas

El paquete `yardstick` nos proporciona multitud de métricas, como se puede ver en [17], y de hecho muchas más de las que llegaremos a usar. Esto se debe, en primer lugar, a que nuestro problema es de regresión y no nos interesan estadísticos como la especificidad o sensibilidad, restringiéndonos al conjunto de medidas aplicables a los modelos de regresión. En segundo lugar, queremos restringirnos a métricas que no solo sean de fácil interpretación, sino que también sean apropiadas para el conjunto de datos considerado y el objetivo que buscamos: predecir *average*.

Con esta idea en mente, seleccionamos las métricas *rmse*, el error cuadrático medio; *rsq*, el R^2 *y mae*, el error absoluto promedio. Los dos primeros son elecciones habituales dentro de cualquier estudio que incluya un modelo de regresión, y el último es un añadido específico para nuestros intereses, pues nos permite saber cuánto podemos llegar a desviarnos del valor real en términos absolutos.

6.2.2 El modelo lineal como predictor

En esta sección nos centraremos en la construcción de un modelo lineal que logre predecir el valor de *average* a partir de las otras variables, con el objetivo de o bien completar valores perdidos de la puntuación o bien, y este es el interés principal, saber hasta qué punto nuestras variables predictoras explican el comportamiento de la variable dependiente. En la siguiente sección nos restringiremos a variables que se pueden estimar o conocer a priori, y aprovecharemos los resultados obtenidos en esta como referencia, para saber hasta qué punto dichas variables son relevantes en nuestro modelo.

Siguiendo la idea comentada, como primera aproximación planteamos recurrir al modelo lineal con los datos ya obtenidos, siguiendo los pasos recomendados por `tidymodels`. Es por ello que, primeramente, definimos el modelo con el que trabajaremos, indicando que es de regresión y que utilizará el sistema *lm*, el básico de R. A continuación, realizamos el ajuste con la orden *last_fit* de tune, que determina el mejor modelo con el conjunto de entrenamiento y lo evalúa en el conjunto test.

```
lm_spec =
  linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")
lm_mod = my_wf_lm %>%
  last_fit(split = bgg_games_split,
           metrics = metric_set(rmse, rsq, mae, mpe))
```

Figura 6.9: Especificación del modelo y ajuste con *last_fit* sobre *my_wf_lm*, adaptación de *my_wf* para este modelo.

Fuente: Elaboración propia.

Una vez hemos ajustado nuestros datos, podemos obtener las métricas mencionadas en la sección anterior con la orden *collect_metrics*, siendo el resultado de la misma:

.metric <chr>	.estimate <dbl>
rmse	2.1484796
rsq	0.2441945
mae	1.4186363

.metric <chr>	mean <dbl>
rmse	2.1014154
rsq	0.2611624
mae	1.3863723

Figura 6.10: Métricas obtenidas con el modelo lineal estimadas por validación cruzada con uno y 10 pliegues, respectivamente.

Fuente: Elaboración propia

En este caso, aunque el error cuadrático medio pudiera parecer reducido, dado que estamos trabajando con valores que varían entre 1 y 10, supone una desviación importante respecto de la realidad, lo que queda representado en el R^2, que nos indica que explicamos menos del 30% de los valores de los resultados obtenidos y, de hecho, si vemos el error absoluto medio, observamos que nuestra esperanza es tener un error absoluto de aproximadamente 1.4 puntos, lo que supone un rango de variación de casi 3 puntos. Cabe destacar que la estimación con un pliegue se asemeja mucho a la realizada con 10 pliegues, lo que valida los valores del primero.

Esta variación, la explicada por el *mae*, nos da una idea, y es la de hacer una comprobación adicional, pues dicha métrica nos podría estar indicando que estamos prediciendo valores mayores que la transformación asociada al 10 o menores que la asignada al 1, lo que resulta absurdo. Es por ello que planteamos el modelo lineal truncado, en el cual almacenamos los valores máximo y mínimo de *average* en el conjunto de entrenamiento, y procedemos como sigue a la hora de realizar las predicciones:

```
lm_mod_fit_trunc = lm_mod_fit %>%
  predict(lm_new_data) %>%
  mutate(.pred = ifelse(.pred < min_average, min_average, .pred)) %>%
  mutate(.pred = ifelse(.pred > max_average, max_average, .pred)) %>%
  bind_cols(truth = lm_new_data$average)
```

Figura 6.11: Ajuste con el modelo lineal truncado.

Fuente: Elaboración propia

Con este, ejecutando las órdenes asociadas a las métricas antes mencionadas, obtenemos los siguientes resultados:

.metric <chr>	.estimate <dbl>
rmse	2.1412793
rsq	0.2589328
mae	1.4167652

Figura 6.12: Métricas del modelo lineal truncado con validación cruzada de un pliegue.

Fuente: Elaboración propia

Como podemos ver, no se produce prácticamente mejora alguna, siendo en todo caso insuficiente como para que merezca la pena plantearse este problema frente al otro, pues este incluye dos comprobaciones por cada fila del conjunto test, llegando a ejecutarse en un tiempo considerable si trabajamos con un gran conjunto de datos.

Dejando a un lado el modelo truncado y volviendo al modelo lineal general, para finalizar esta sección, veamos cómo este ha predicho algunas entradas, para comparar este resultado con el de las métricas obtenidas, y acompañemos estos resultados de los mayores coeficientes obtenidos dentro del modelo, por tipo de variable:

.pred <dbl>	average <dbl>
6.005043	6.12209
5.682880	6.75402
6.960323	5.93088
7.150017	7.09789
5.857773	5.28925
6.270949	6.84385

Figura 6.13: Predicciones con el modelo lineal.

Fuente: Elaboración propia

Observamos grandes diferencias entre estos primeros datos, llegando a predecir el modelo más de un punto menos del valor real, como se puede observar en la segunda línea o en la tercera línea.

Siguiendo con el modelo lineal general, veamos ahora los coeficientes de las variables sobre las que nos hemos construido variables *dummy* adicionales, ordenados por valor decreciente de valor absoluto:

Variable <chr>	Coefficient <dbl>
wanting	0.2971182901
owned	0.2896226623
averageweight	0.2802449936
usersrated	-0.2505267282
wishing	-0.1951987330
trading	-0.0755904908
numweights	0.0558479419
numplayers_best	0.0478956511
minplayers	0.0411827911
numplayers_recommended	0.0276832327

Figura 6.14: Coeficientes con el modelo lineal, primera parte.

Fuente: Elaboración propia

A la vista de la Figura 6.14, parece que las variables que más efecto tiene en el modelo son las relativas al movimiento del juego dentro de la plataforma, así como la complejidad media, distinguiéndose considerablemente de los otros coeficientes en cuanto a valor absoluto, correspondiéndose esto con lo que ya nos esperábamos tras el análisis de la correlación de la sección 4.2. De hecho, parece que la dificultad es algo que se valora positivamente, pues con este modelo podemos afirmar, que al menos en un número importante de casos, incrementos en la complejidad tendrán asociados crecimientos en la valoración media. Tal y como era de esperar en este caso, las variables *wanting, wishing* y *owned* influyen en gran medida en el modelo, pues son claramente indicadores del comportamiento del mercado para un juego concreto, como mencionábamos en la sección 3.1.2, si bien solo *wanting* y *owned* logran incrementar la puntuación, reduciéndose en caso contrario. Sin embargo, es destacable remarcar que el número de jugadores que tienen el juego en su lista de deseados influye negativamente, en contra de lo que podríamos esperar, un comportamiento semejante al de la variable *wanting*, esto es, que existe una relación clara entre gente que quiere

un juego y la puntuación que tiene el mismo, asumiendo que la gente busca juegos "buenos".

Variable <chr>	Coefficient <dbl>
minplaytime	-0.0229783094
maxplaytime	0.0203873819
yearpublished	0.0119626742
numplayers_not_recommended	0.0098202156
numcomments	0.0097573951
sentiment_mean	-0.0047882150
maxplayers	0.0009515515

Figura 6.15: Coeficientes con el modelo lineal, segunda parte.

Fuente: Elaboración propia

En relación con esto último, la variable *trading* tiene una interpretación semejante pero inversa, y es que si hay gente ofreciéndolo podemos asumir que esto ha de deberse a que el juego no es demasiado bueno, siendo esperable por lo tanto una puntuación menor.

Otra información interesante que podríamos obtener en un primer momento es que el ser un juego especialmente pensado para muchos jugadores, es decir, con un valor elevado de *numplayers_best* y/o de *numplayers_recommended*, mejora la calidad del juego, lo que explicaría también en parte los resultados sobre frecuencias de juegos de tipo *party* en la sección 4.2.3.

Por último, podríamos explicar el signo positivo en *numweigths*, por ejemplo, pensando que aquellos que consideran un juego como desafiante o especialmente entretenido acuden con rapidez a la página para asignarle una complejidad, y aprovechan la visita para asignarle una puntuación, lo que daría también una explicación de la correlación vista en la sección 4.2. Para poder contrastar esta hipótesis, sin embargo, sería necesario recurrir a información adicional de la que no disponemos, como el tiempo transcurrido entre una valoración del juego y de su dificultad, y en todo caso también el sentimiento asociado al comentario de existir este, pues podría explicar si realmente el usuario ha recibido o no una mala impresión. Siguiendo con este punto, el signo negativo de *usersrated* contrasta con este último, el positivo de *numweights*, y es que lo esperable sería que tuviesen valores semejantes, especialmente al tener en cuenta la correlación obtenida en la sección 4.2. Como decíamos, para poder explicar este comportamiento necesitamos disponer de información adicional, en un primer momento.

Variable <chr>	Coefficient <dbl>
`category:Abstract Strategy`TRUE	0.174492296
`category:Card Game`TRUE	-0.075611876
`category:Children's Game`TRUE	-0.164417909
`family:Players: Games with Solitaire Rules`TRUE	0.087418766
`family:Players: Two Player Only Games`TRUE	-0.011428294
`family:Series: Monopoly-Like`TRUE	-0.713457135
`mechanic:Hand Management`TRUE	0.138923990
`mechanic:Hexagon Grid`TRUE	-0.001722348

Figura 6.16: Coeficientes con el modelo lineal con algunas variables dummy.

Fuente: Elaboración propia

Para las tres categorías principales, *boardgamecategory, boardgamefamily* y *boardgamemechanic*, como no son mutuamente excluyentes, el análisis de los coeficientes es en términos absolutos, pues se pueden acumular por cómo hemos construido las variables *dummy* asociadas. Así, por ejemplo, si un juego es de estrategia abstracta y de cartas, el resultado se incrementaría por una parte y se reduciría por la otra, obteniendo de esta forma una puntuación mayor en 0.1 puntos. Hay que tener cuidado, pues con toda seguridad nuestro modelo padece problemas de colinealidad al haber una fuerte correlación entre las variables, perteneciendo en muchos casos un juego a una categoría al no pertenecer a otras.

Por otro lado, destacamos la familia, la característica de ser un juego con reglas para jugar en solitario, condición que no hubiésemos considerado tan relevante en un primer momento, pero que a la vista de los resultados podemos afirmar que influye favorablemente en los resultados. Sin embargo, el punto más llamativo es lo mucho que el pertenecer a la familia de juegos semejantes al *Monopoly* afecta negativamente a la puntuación obtenida, pudiendo llegar a bajarla casi en un punto.

Por último, para las variables categóricas que sí que eran mutuamente excluyentes, asociando la edad recomendada de más de 21 años y el ser injugable en otro idioma a la media, nos encontramos con los siguientes resultados:

Variable <chr>	Coefficient <dbl>
`language_dependence:Extensive use of text`TRUE	-0.02605732
`language_dependence:Moderate in-game text `TRUE	-0.16864070
`language_dependence:No necessary in-game text`TRUE	-0.12102063
`suggested_playerage:12`TRUE	0.45505010
`suggested_playerage:14`TRUE	0.40033201
`suggested_playerage:16`TRUE	0.24105074
`suggested_playerage:18`TRUE	0.06721657

Figura 6.17: Coeficientes con el modelo lineal con las variables *dummy* mutuamente excluyentes.

Fuente: Elaboración propia

En este caso sí que debemos trabajar de forma relativa, afirmando que, por ejemplo, el uso reducido de texto reduce la puntuación esperada de un juego. Tal y como veremos en la siguiente sección, esto podría deberse al sobreajuste, pues no se corresponde ni con nuestra intuición ni con los resultados obtenidos hasta ahora. Por otro lado, vemos que la edad mínima recomendada afecta en general incrementando la puntuación esperada al ser la edad mínima menor, lo que se correspondería con la gráfica de frecuencias de la sección 4.2.1, en la que se favorecía a los juegos para adolescentes y preadolescentes frente a aquellos pensados para un público adulto. Este comportamiento será analizado, más en detalle, en la siguiente sección, al ser un patrón inesperado.

6.2.3 Aplicación: claves del éxito

En este punto buscamos ser capaces de distinguir las variables que más influyen dentro de las que sabemos que se pueden medir o estimar sin haber lanzado el producto, de forma tal que podamos analizar, de manera específica, los valores que han de tomar las variables seleccionadas para maximizar la puntuación posible, esto es, que logren un mejor éxito en el mercado.

Por lo tanto, descartamos las siguientes variables, al no poder estimarse a priori en casi ninguna situación: *usersrated, owned, trading, wanting, wishing, numcomments* y *numweights*. Suponemos que otras variables como *averageweight* o *sentiment* pueden estimarse con un grupo de prueba previo sin mucha dificultad, con una encuesta tras unas partidas, para un público lo suficientemente variado. Almacenamos estas variables en un vector llamado *unknown* y actualizamos nuestro flujo de trabajo, de forma tal que

se deseleccionen estas variables a la hora de preprocesar los datos, incluyendo por lo tanto este paso en la receta del modelo.

```
lm_rec = my_wf %>%
pull_workflow_preprocessor() %>%
  step_select(-all_of(unknown))
my_wf_lm = my_wf %>%
  update_recipe(lm_rec) %>%
  add_model(lm_spec)
```

.metric <chr>	mean <dbl>
rmse	2.1025152
rsq	0.2626257
mae	1.3957590

Figura 6.18: Actualización del modelo lineal y métricas obtenidas, estimadas por validación cruzada con diez pliegues.

Fuente: Elaboración propia

Ajustando en este caso el modelo, observamos que estas variables desconocidas no ayudaban en gran medida a estabilizar el error, pues hemos obtenido resultados casi idénticos a los anteriores. Esto, por supuesto, puede deberse también a que solo hemos suprimido 7 variables de las más de 50 con las que estábamos trabajando, y es que podríamos estar sobreajustando los datos con dicho número de campos, lo que explicaría también el gran valor obtenido para el error cuadrático medio o para el error absoluto medio.

Siguiendo como antes, representamos para las variables numéricas los coeficientes asociados en este modelo, buscando explicaciones semejantes a las ya obtenidas:

Variable <chr>	Coefficient <dbl>
averageweight	2.967497e-01
numplayers_recommended	4.553391e-02
numplayers_best	4.196355e-02
minplayers	3.956162e-02
minplaytime	-2.759054e-02
maxplaytime	2.755423e-02
numplayers_not_recommended	2.561593e-02
sentiment_mean	1.427592e-02
yearpublished	1.250491e-02
maxplayers	-5.803187e-05

Figura 6.19: Coeficientes en el nuevo modelo lineal, para variables numéricas.

Fuente: Elaboración propia

En este caso obtenemos resultados ligeramente diferentes, y es que aunque la variable con más peso sea *averageweight*, en segundo lugar nos encontramos con *numplayers_recommended*, que junto con *numplayers_best* nos indican que es preferible tener un gran número de jugadores como mejor opción para el juego, independientemente del número teórico de jugadores. Es llamativo ver que, junto con estas variables, todas salvo *minplaytime* y *maxplayers* influyen favorablemente en la valoración de un juego, siendo por lo tanto recomendable tener un número de jugadores máximo elevado, y con una duración mínima tan reducida como sea posible.

En concreto, nos llama la atención ver cuál es la variable que más debemos reducir para obtener un resultado favorable: la duración mínima de la partida. Aparentemente, se busca que, aunque un juego pueda alargarse bastante, una partida corta tenga una duración especialmente reducida. Esto, por otro lado, parece ser incompatible en la mayor parte de los casos con la complejidad del juego, lo que podría explicar también en parte el resultado relativo a la correlación entre estas dos variables, obtenido en la sección 4.2.

De hecho, viendo la Figura 6.19 y combinándola con los resultados obtenidos en la sección 4.2.3, da la impresión de que el mercado le da un gran peso a los juegos de ese tipo, juegos de duración limitada, pero a la par lo suficientemente complejos y para un número elevado de jugadores, siempre que asociemos dicho peso en el sector a la puntuación que recibe el juego mediante diferentes valoraciones.

Si nos centramos ahora en las variables categóricas, obtenemos resultados diferentes a los de la Figura 6.19, y es que el efecto de varios niveles se ha modificado, de una forma más o menos considerable, en términos de los coeficientes asociados. Aparentemente, la categoría para juegos semejantes al *Monopoly* no tiene una buena recepción, y por lo tanto debería ser algo a evitar a la hora de diseñar un juego. Igualmente, las mecánicas de “girar o lanzar y mover” y de *Trivial* tampoco están asociadas a un gran éxito entre el público, si bien para poder interpretar más este resultado sería necesaria más información sobre la categoría.

Variable <chr>	Coefficient <dbl>
`family:Series: Monopoly-Like`TRUE	-0.7625238
`category:Wargame`TRUE	0.5597832
`mechanic:Roll / Spin and Move`TRUE	-0.3385415
`category:Trivia`TRUE	-0.2811190
`family:Sports: Baseball`TRUE	0.2615813
`category:Educational`TRUE	-0.2467644
`family:Brands: Disney`TRUE	-0.2060962
`mechanic:Hand Management`TRUE	0.2060842
`category:Movies / TV / Radio theme`TRUE	-0.2048945
`category:Other category`TRUE	0.1882334

Figura 6.20: Las 10 variables dummy asignadas a *boardgamecategory, boardgamefamily* y *boardgamemechanic* que más efecto tienen en el resultado.

Fuente: Elaboración propia

Por otro lado, vemos que es favorable el dar forma a un juego que se pueda catalogar como *wargame*, siendo esta variable la que más incrementa el éxito del resultado, al menos aparentemente. Como antes, estas variables se pueden sumar entre sí al no ser mutuamente excluyentes, lo que nos indicaría que, por ejemplo, un *wargame* que tenga relación con el *béisbol*, y que incluye alguna mecánica de gestión de recursos en mano, habría de ser un diseño bien recibido, al menos guiándonos por los resultados proporcionados por este modelo.

De esta forma, filtrando las variables que nos puedan interesar o pudiésemos querer incluir en un nuevo juego, considerando los coeficientes podríamos hacernos una idea sobre lo recomendable que podría ser incluir dichas mecánicas o temáticas.

Tocando ahora otro punto, como las variables de edad y dependencia del lenguaje sí que son excluyentes, podemos construir tablas que nos expliquen el comportamiento de los datos dentro de dicha categoría, relativos a la edad recomendada de 21 o más años y al ser imposible de jugar en otro idioma.

Variable <chr>	Coefficient <dbl>
`suggested_playerage:10`TRUE	0.54651520
`suggested_playerage:12`TRUE	0.51367690
`suggested_playerage:14`TRUE	0.46277424
`suggested_playerage:8`TRUE	0.46061335
`suggested_playerage:4`TRUE	0.36785821
`suggested_playerage:6`TRUE	0.32918174
`suggested_playerage:5`TRUE	0.32469511
`suggested_playerage:Unknown`TRUE	0.32264451
`suggested_playerage:2`TRUE	0.28248487
`suggested_playerage:16`TRUE	0.23569417

Figura 6.21: Coeficientes para las variables dummy hijas de *suggested_playerage*.

Fuente: Elaboración propia

Con esta gráfica podemos ver que, en general, el apostar por juegos con una edad reducida es mejor que desarrollar un juego que tenga una edad mínima recomendad de 21 o más años, y cuanto menor la edad, mejor el resultado obtenido, hasta los 10 años, momento a partir del cual se empieza a reducir nuevamente el éxito esperado. Este resultado es coherente con el obtenido en la sección anterior, lo que nos hace pensar que, al menos con estas variables. el modelo no está sobreajustando los datos.

Variable <chr>	Coefficient <dbl>
`language_dependence:Unknown`TRUE	-0.17279736
`language_dependence:Moderate in-game text `TRUE	-0.12094467
`language_dependence:Some necessary text`TRUE	-0.08754246
`language_dependence:No necessary in-game text`TRUE	-0.08060307
`language_dependence:Extensive use of text`TRUE	0.02098877

Figura 6.22: Coeficientes para las variables dummy hijas de *langugage_dependence*.

Fuente: Elaboración propia

Por último, nos encontramos nuevamente con un resultado inesperado, y es que aparentemente el reducir el peso del texto en el juego reduce la puntuación del mismo, siendo de esta forma la conclusión que el diseñar un juego con una gran dependencia del lenguaje mejora la calidad del mismo. Esto se podría explicar basándonos en que de esta forma es posible incluir mucha más información en tableros y cartas, obteniéndose así juegos más complejos, con más variedad de opciones, a priori, que otros que no se apoyan en texto alguno, y que se ven obligados a incluir multitud de ilustraciones como sustitutos, ocupando más espacio que los textos, en general.

Con esto hemos logrado obtener algunas posibles claves en lo referido a obtener cierto éxito en el mercado de los juegos de mesa, si bien hay que tener en cuenta que el elevado error asociado a nuestro modelo ha de ser tenido en cuenta, pues esto habría de servir, en todo caso, como una guía, siendo imperativo de cualquier forma el llevar a cabo pruebas para estimar las variables mencionadas y, sobre todo, para obtener valoraciones de los usuarios.

6.2.4 Regresión KNN

Completamos esta sección y la siguiente modificando el modelo con variables desconocidas, viendo si podemos obtener otros resultados de regresión mejores al considerar KNN y árboles de regresión, modelos que como ya mencionamos no tienen apenas hipótesis sobre los datos, y que en multitud de ocasiones pueden producir buenos resultados.

.metric <chr>	mean <dbl>
rmse	2.218919
rsq	0.129693
mae	1.505472

Figura 6.23: Métricas estimadas por validación cruzada con 10 pliegues, para las variables conocidas.

Fuente: Elaboración propia

Comparando los resultados presentes en esta figura con los que vimos en la Figura 6.18, vemos que no solo no hemos mejorado nuestras estimaciones, sino que hemos incrementado el valor de todas las métricas de error.

Esto nos permite afirmar que, al menos para este conjunto de datos, el modelo lineal presenta un comportamiento menos sesgado que el KNN, y por lo tanto es preferible a este último. Quizás, añadiendo los rangos dentro de diferentes clasificaciones, medida al fin y al cabo de cercanía, podríamos mejorar estos resultados, pero esto nos sería de poca utilidad, pues resulta difícil estimar una clasificación a priori, especialmente para un juego que aún no se encuentra en el mercado.

6.2.5 Árboles de regresión

Siguiendo con la idea de la anterior sección, intentamos construir un modelo de árbol de regresión para ver si de esta forma logramos reducir el error, obteniendo de esta forma mejores predicciones.

.metric <chr>	mean <dbl>
rmse	2.151422
rsq	0.222636
mae	1.443057

Figura 6.24: Métricas estimadas por validación cruzada con 10 pliegues, para las variables conocidas.

Fuente: Elaboración propia

Obtenemos resultados semejantes al modelo lineal, sacrificando ligeramente la precisión, como podemos ver al incrementarse el error cuadrático medio y al reducirse el R^2. Sin embargo, si observamos el árbol obtenido en la Figura 6.25, nos encontramos con que con un árbol tan sencillo, basado únicamente en las variables *averageweight, mechanic:Roll / Spin and Move* y *category:wargame*, con 5 nodos únicamente, logramos resultados semejantes a los del modelo lineal.

```
node), split, n, deviance, yval
      * denotes terminal node

 1) root 19432 35826.410 5.615857
   2) averageweight< 0.4102185 11984 22369.100 5.190890
     4) mechanic:Roll / Spin and Move>=0.5 3059  5837.405 4.770974 *
     5) mechanic:Roll / Spin and Move< 0.5 8925 15807.430 5.334814
      10) category:Wargame< 0.5 8000 13586.870 5.257176 *
      11) category:Wargame>=0.5 925  1755.286 6.006281 *
   3) averageweight>=0.4102185 7448  7810.673 6.299638
     6) averageweight< 0.6949587 2603  2909.190 5.831450 *
     7) averageweight>=0.6949587 4845  4024.363 6.551173 *
```

Figura 6.25: Representación del árbol obtenido.

Fuente: Elaboración propia

Era esperable que el algoritmo optase por la complejidad media frente a otras variables, debido en parte a la gran correlación que ya vimos que existía entre ellas. Por otro lado, este modelo nos indica que el mejor resultado lo conseguimos para los juegos más complejos, correspondiéndose con el resultado obtenido en los modelos de las secciones 6.2.2 y 6.2.3, en las que ya podíamos ver la importancia de esta variable.

En definitiva, con esto hemos obtenido un procedimiento que, si bien menos preciso, nos permite hacer una estimación con increíble rapidez, y que además nos llama la atención sobre el valor que tienen el tener la mecánica “lanzar/girar y mover” y el ser un *wargame*, encajando también estos resultados con los que logramos en la sección anterior.

Destacamos, igualmente, que con este modelo solo predecimos valores entre los 4 y los 6.5 puntos, siendo por lo tanto esperable el gran error cuadrático que hemos obtenido, a pesar de que la media de *average* se encuentre en dicho intervalo.

```
rf_spec = rand_forest() %>%
  set_engine("ranger") %>%
  set_mode("regression")
```

Figura 6.26: Declaración de la especificación del modelo de *random forest.*

Fuente: Elaboración propia

Una forma de extender estos resultados, ya que hemos visto que no son demasiado precisos, sería recurriendo a un modelo de *random forest*. Dado que dentro del paquete `parnsnip` disponemos del sistema *Ranger*, podemos implementar el modelo como se observa en la Figura 6.26 e incorporarlo a nuestro flujo de trabajo con facilidad, siendo nuestros estimadores del error por validación cruzada los que se pueden observar en Figura 6.27.

.metric <chr>	mean <dbl>
rmse	2.1215880
rsq	0.2455888
mae	1.3760448

Figura 6.27: Estimación del error por validación cruzada de 10 pligues con el modelo de *random forest*.

Fuente: Elaboración propia

Así, observamos que el resultado obtenido de esta forma no dista demasiado del que logramos con un árbol de regresión habitual, cuando podríamos haber esperado una mejora significativa con este proceder. Por lo tanto, podríamos pensar que el problema en estos modelos se encuentra en los datos con los que trabajamos, y no tanto en la selección del modelo.

Si se diese dicha situación, en la que en los datos hubiese mucho ruido, quizás sería conveniente recurrir a un modelo de redes neuronales, que como sabemos es robusto ante dichas interferencias, pudiendo proporcionarnos mejores resultados que los vistos hasta ahora.

6.2.6 Predicción con redes neuronales

Aunque nuestro objetivo en esta sección sea, en un primer momento, identificar los puntos más influyentes a la hora de hora de diseñar un juego, teniendo en cuenta factores como la duración, el número de jugadores recomendado o la complejidad, es cierto que podemos también intentar construir un modelo de predicción recurriendo a redes neuronales, para ver hasta qué punto somos capaces de predecir los resultados con las variables dadas, dejando a un lado la interpretabilidad para favorecer la obtención de mejores predicciones.

```
form2 <- train_data %>% select(-average) %>%
  colnames %>% paste0(collapse = " + ")
my_formula <- "average ~ " %>%
  paste0(form2, collapse = "") %>% as.formula()
my_net <- neuralnet(formula = my_formula,
                    data = train_data,
                    hidden = 3,
                    threshold = 0.3)
```

```
      error reached.threshold             steps
1.16934e+04       2.96649e-01       1.49350e+04
```

Figura 6.28: Declaración de la red neuronal *my_net*, con 3 neuronas ocultas y un umbral de 0.3 para asegurar la convergencia temprana. Fragmento del resultado.

Fuente: Elaboración propia

Recurriendo al paquete `neuralnet`, como se ve en la Figura 6.28, somos capaces de definir la red neuronal *my_net*, trabajando con el objeto *train_data*, construido a partir de nuestro *split* inicial y de la orden *prep* del paquete `recipes`, para poder aplicarle la receta que ya diseñamos previamente. Siguiendo este proceder, somos capaces de generar nuestra red neuronal, aceptando un error algo mayor de lo habitual para asegurar una convergencia temprana, en pocos pasos, siendo el resultado el que se puede observar en la Figura 6.29, alcanzado tras 14.935 iteraciones del algoritmo de mejora de los pesos.

A continuación, aprovechamos un objeto que definimos al mismo tiempo que nuestro flujo de trabajo, *bake_res*, que no era más que el resultado de procesar el conjunto test a través de la receta mencionada. De esta forma, obtenemos las siguientes métricas para esta red neuronal:

.metric <chr>	.estimate <dbl>
rmse	2.041934
rsq	0.296564
mae	1.339938

Figura 6.29: Errores obtenidos con *my_net* y el conjunto *bake_res*.

Fuente: Elaboración propia

A la vista de este resultado, a pesar de haber aceptado un mayor error, reafirmamos lo comentado en la anterior sección, y es que parece que no se puede extraer mucha más información de los datos proporcionados, no sin mejorar nuestros modelos, ya sea incrementando el tamaño muestral, el número de iteraciones o reduciendo, por ejemplo, el umbral de error a la hora de definir la red neuronal.

De cualquier forma, cabe destacar que se puede observar una mejora tanto en el R^2 como en el error absoluto medio, lo que nos puede dar la idea de que el seguir por este camino podría proporcionarnos buenos resultados.

6.3 Inferencia: influencia de la temática

Buscamos ahora, siguiendo algunas de las ideas vistas a lo largo del proyecto, centrarnos en la importancia de la temática de los juegos, esto es, el valor del campo *boardgamecategory*. Ya vimos en varios resultados, primeramente en el análisis gráfico en la sección 4.2.1, y más adelante en la construcción de modelos en las secciones 6.2.2 y 6.2.3 que este era una variable que condicionaba el resultado asociado al juego, esto es, el promedio de su puntuación.

Para analizar esto con cada una de las categorías, para todos los juegos disponibles en la base de datos, leemos los datos y ejecutamos un análisis ANOVA para cada categoría, previa creación de las variables *dummy* asociadas, y guardamos la información en un *tibble*. Podemos ver ahora, en la siguiente tabla, un filtro sobre dicho *tibble*, seleccionando únicamente aquellas categorías para las que el test nos indique que existe evidencia de que influyen en el resultado.

category <chr>	pvalue <dbl>	coefficient <dbl>
Expansion.for.Base.game	6.377913e-257	1.9431241
Age.of.Reason	4.602574e-21	1.7132994
Wargame	0.000000e+00	1.6806019
Post.Napoleonic	1.933593e-24	1.5991274
Book	1.434021e-50	1.5969182
World.War.II	1.853964e-157	1.5568813
Educational	4.955458e-193	-1.5517746
Napoleonic	1.351877e-39	1.5331180
Religious	1.728800e-34	-1.4987997
Civil.War	1.181775e-17	1.4709067

Figura 6.30: Selección de categorías con p-valores significativos y coeficientes asociados.

Fuente: Elaboración propia

A la vista del resultado, podemos afirmar que las expansiones para juegos previos tienen una buena recepción por lo general, llegando a incrementar hasta en 2 puntos la puntuación frente a otras posibles categorías. Como en la sección anterior, vemos que los *wargames* son una buena opción a la hora de escoger una temática para un juego, y no solo eso, sino que si nos fijamos, la mayor parte de las temáticas aquí presentes, con coeficientes positivos, tienen una relación clara con el concepto de *wargame*, lo que nos hace pensar que los *wargames* históricos son recibidos de forma muy positiva por el público.

Por último, nos llama la atención ver que, como observamos en la sección 6.2.3, los juegos educacionales no tienen éxito, y de hecho reducen considerablemente las expectativas de popularidad frente a cualquier otra categoría o temática. De forma semejante, la temática religiosa tampoco tiene una buena recepción, y es por lo tanto algo a evitar si buscamos el mayor éxito posible.

6.4 Series temporales aplicadas a la familia Kickstarter

Siguiendo en la línea del análisis descriptivo, recordemos que habíamos tomado un subconjunto de datos, con entradas para cada año seleccionado, para cada categoría, y habíamos obtenido los gráficos de la Figura 4.28, en la que habíamos visto un comportamiento creciente generalizado para todas las categorías para las que contábamos con entradas en los últimos 20 años.

Teniendo esto en cuenta, y dado que nuestro objetivo es obtener tantas conclusiones como sea posible sobre el sector, nos restringimos a la categoría "*Crowdfunding:*

Kickstarter", esto es, aquellos juegos que han sido producidos a través de una campaña de mecenazgo en *kickstarter.com*, la plataforma más importante a nivel mundial de este tipo [34]. Estos resultados podrían ser combinados con los obtenidos en la sección 6.2 para conocer la viabilidad de un proyecto, lanzándolo a través del portal mencionada.

Por lo tanto, buscamos ahora analizar el comportamiento de las valoraciones medias a lo largo de los años para los juegos de *kickstarter.com*, dado que la relevancia de poder obtener resultados al respecto ha quedado clara.

```
bgg_games_fam = bgg_games %>%
  select(average, yearpublished, boardgamefamily) %>%
  filter(yearpublished > 1999 & yearpublished < 2020) %>%
  filter(str_detect(
    paste0(unlist(boardgamefamily), collapse = ""),
    pattern = "Crowdfunding: Kickstarter")) %>%
  select(-boardgamefamily)
```

Figura 6.31: Preparación de los datos mediante sucesivos filtros.

Fuente: Elaboración propia

Primeramente, destacamos la forma en la que hemos seleccionado las entradas de esta categoría, pues trabajamos con todas las entradas disponibles y el recurrir a la orden *unnest_longer* sobre la variable *boardgamecategory* incrementa la dificultad de esta orden, pues triplica prácticamente la longitud de los datos, llegando de esta forma a trabajar con 900.000 entradas si seguimos ese camino. Es por ello que hemos optado por filtrar primeramente por columnas y por filas, para luego simplemente detectar el patrón asociado a la categoría dentro de cada lista, previamente convertida en vector con la orden *unlist*. Con esto, el resultado se obtiene de una forma más rápida.

Sin embargo, cabe destacar que no podíamos aplicar esta estrategia en los capítulos anteriores, pues estábamos interesados en trabajar con todas las categorías, familias y mecánicas, mientras que el código expuesto en la figura solo nos permite obtener un subconjunto de las mismas.

Una vez ejecutada la orden obtenemos el conjunto *bgg_games_fam*, en el que almacenamos únicamente el año y la puntuación de los juegos de nuestra base de datos que tengan entre sus familias el haber sido lanzados a través de *kickstarter.com.* Obtenemos ahora nuevas gráficas que complementan a las obtenidas en la sección 4.2.2, ofreciendo información adicional.

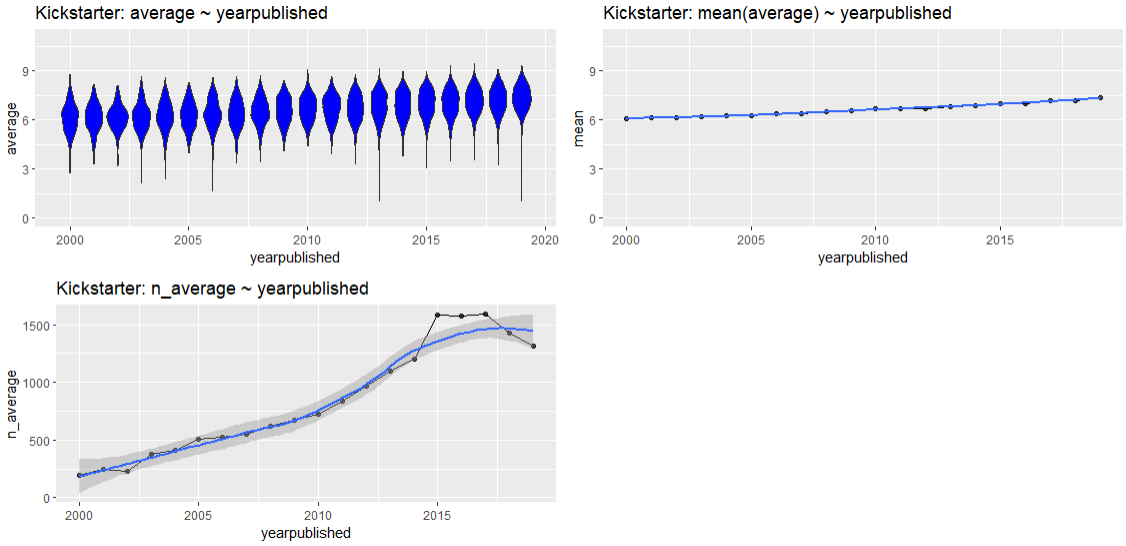

Figura 6.32: Comportamiento de *average* y número de entradas para juegos de Kickstarter, 2001-2019.

Fuente: Elaboración propia

Podemos observar en estas gráficas que existe una gran variabilidad en los juegos que han sido lanzados a través de la plataforma mencionada, encontrándonos con entradas que varían entre la puntuación 3 y 9, aunque la mayor concentración se da alrededor del 6, donde recordemos que se encontraba la media de los juegos de la BGG.

Centrándonos ahora en la gráfica inferior de la Figura 6.32, vemos que el número de juegos producidos mediante esta página de mecenazgo se ha incrementado a lo largo de los últimos años, y mediante el método loess estimamos que seguirá creciendo, si bien experimentando una bajada en la pendiente.

Por otro lado, aunque sea suave, es cierto que podemos observar cierta tendencia positiva en la media de los juegos de esta categoría, pasando de puntuaciones de 6 a prácticamente puntuaciones de 8. Con este conjunto de datos, la puntuación media a lo largo de los años, es con el que planeamos trabajar.

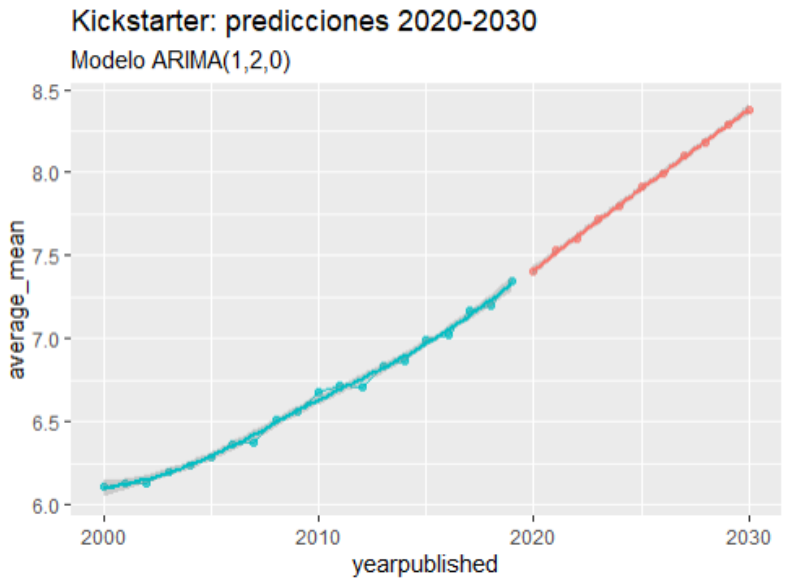

Figura 6.33: Predicciones con el modelo ARIMA(1,2,0), generado de forma automática.

Fuente: Elaboración propia

Recurriendo a un ajuste automático de modelo de series temporales ARIMA(*p,d,q*) obtenemos como modelo óptimo el ARIMA(1,2,0), indicándonos que es una serie integrada de segundo orden, con una parte autorregresiva de orden uno, indicándonos que cada año depende del anterior, y que existe una tendencia intrínseca, no nula. De esta forma, construyendo predicciones para los siguientes 10 años, aunque tengamos una muestra pequeña, nos da una idea del comportamiento que podríamos esperar para los juegos lanzados mediante esta plataforma en este futuro, indicándonos que el recurrir a ella puede ser una buena estrategia para mejorar los posibles resultados que se obtengan.

7 CONCLUSIONES

Dada la extensión del proyecto, resulta fundamental el disponer de un capítulo dedicado únicamente a la recopilación de los diferentes logros alcanzados a lo largo del trabajo- Incluimos en este punto, por lo tanto, un desglose de los principales resultados que hemos obtenido a lo largo del trabajo, con el objetivo de aclarar sobre qué puntos hemos conseguido extraer información adicional a la ya conocida, y a qué preguntas hemos conseguido dar respuesta, así como la forma en la que hemos obtenido dicho resultado.

7.1 Conclusiones del trabajo

A lo largo del trabajo hemos recurrido a multitud de herramientas y procedimientos, partiendo primeramente del procesamiento y preparación de los datos, para luego aplicar diferentes estrategias, desde el análisis gráfico y descriptivo a la modelización y el procesamiento del lenguaje natural. Con estos pasos hemos logrado obtener información útil relativa al mercado de los juegos de mesa, como qué temáticas, categorías y mecánicas son las que más éxito pueden proporcionar, como ya vimos; o cuáles son los factores a los que debemos dedicarles una mayor atención a la hora de buscar producir un juego popular, como la complejidad, que como hemos visto es algo fundamental, habiendo estudiado estos puntos a través del modelo lineal y de un árbol de regresión en la sección. Sin embargo, estos resultados han llevado asociado un error significativo en varios casos, como en los modelos predictivos, y es por ello vital que, a la hora de tener en cuenta la información proporcionada, se consideren también otros factores, llevando a cabo un análisis del mercado específico para el juego en cuestión.

Siguiendo con el punto de las estimaciones, basándonos en los resultados logrados hemos podido comprobar que el mercado de los juegos de mesa, tal y como se comentaba al principio del proyecto, se encuentra en auge, recibiendo cada vez mejores críticas los juegos que se lanzan cada año. Esto, junto con la información obtenida sobre *kickstarter.com*, nos proporciona una perspectiva general, indicándonos que es un sector en el que es posible invertir esfuerzo y obtener un gran beneficio, siempre y cuando se controlen los puntos mencionados.

Por otro lado, hemos logrado proporcionar resúmenes de grandes conjuntos de texto, y no solo hemos conseguido compactar la información contenida en cada comentario a un valor representativo indicando la percepción del público, sino que también hemos producido, de forma gráfica, herramientas que podrían llegar a ser de utilidad aplicadas a la hora de analizar la respuesta de los usuarios, pudiéndose emplear también en el

momento de querer transmitir los puntos clave de un juego. Adicionalmente, hemos conseguido desarrollar otra herramienta cuyo interés consideramos demostrado y que se podría incluir en multitud de plataformas relacionadas con los juegos de mesa, una que nos permite generar una relación de semejanza entre juegos dados únicamente los comentarios asociados a los mismos, pudiendo crear de esta forma una jerarquía que nos permitiría buscar juegos semejantes a uno dado.

Finalmente, hemos obtenido algunos comportamientos particulares e información variada sobre temáticas concretas, aplicando técnicas de minería de texto, así como sobre la dependencia del lenguaje en un juego y su importancia, semejante a la del número de jugadores, que como ya mencionábamos en el capítulo anterior, puede resultar determinante a la hora de evaluar un juego.

Como otro punto que comentar, a lo largo de todo el trabajo hemos hecho un uso extensivo de paquetes de diferentes enfoques, pensados para diferentes objetivos, obteniendo de esta forma una perspectiva aún mayor de la variedad de herramientas que podemos hallar en R, útiles que, como hemos podido comprobar a lo largo del proyecto, pueden combinarse para producir no solo resultados estadísticamente significativos, sino también estructuras adaptadas para cualquier objetivo que podamos buscar, siendo el único limitante la capacidad del dispositivo con el que estemos trabajando. Especialmente destacables son las técnicas vistas dentro del sector del procesamiento del lenguaje natural con los paquetes `tm` y `tidytext`, pues se trataba de un área desconocida, que no habíamos visto a lo largo del grado, y dentro de la cual ahora conocemos varios procedimientos para extraer información útil.

Por otra parte, y siguiendo con la línea de puntos destacados conforme hemos desarrollado el trabajo, el enfrentarnos a un problema real, de dimensiones auténticas, nos ha obligado a modificar la forma en la que abordamos múltiples puntos, teniendo en cuenta que operaciones que podrían tardar menos de un segundo con conjuntos relativamente pequeños en nuestro caso pueden llegar a tardar horas en ejecutarse, siendo por lo tanto imperativo no solo el encontrar una definición eficiente, sino también el construir las funciones de forma tal que obtengamos valores que nos permitan estimar los tiempos de ejecución, así como identificar los campos más problemáticos dentro de una función.

Por último, no solo el tratar con los datos, sino el conocer el funcionamiento de varias plataformas con bases de datos como *kickstarter.com* o *boardgamegeek.com* nos ha permitido tomar consciencia de la importancia de buscar el objetivo de construir bases

de datos funcionales, pero que también estén bien estructuradas y ordenadas, de forma tal que no sea necesario preprocesar cada resultado que podamos obtener.

7.2 Caminos descartados

Es claro que el proyecto que hemos decidido abordar es muy ambicioso, y a lo largo del mismo el esfuerzo tecnológico de desarrollo ha sido muy significativo, acompañado de un continuo análisis estadístico de gran importancia, que nos ha permitido conducir los estudios realizados. Además, otros factores a tener en cuenta han sido la variedad de aspectos nuevos estudiados y aplicados o conocidos pero en los que se ha profundizado mucho, así como la magnitud del problema escogido, inédita para nuestra experiencia. No obstante, dado que el alcance del trabajo debía acotarse en algún momento, nos hemos visto obligados a descartar algunas vías, que sin embargo resulta interesante mencionar.

Primeramente, dado que nuestro objetivo era analizar el mercado de los juegos de mesa, haciendo especial hincapié en el procesamiento del lenguaje natural, una herramienta que se nos podría venir rápidamente a la cabeza es Twitter, donde claramente podríamos disponer de un gran corpus con el que obtener conclusiones sobre el mercado en general, o sobre juegos particulares. Aunque fue uno de los principales puntos que buscábamos abordar en este proyecto, no se nos concedió el acceso completo a la API, esto es, la versión académica, y con la proporcionada, la estándar, solo podíamos hacer búsquedas en los *tweets* de las últimas 2 semanas, lo que dificultaba considerablemente la recopilación y validación de los datos con otras fuentes. Por estas cuestiones, y al incluir la BGG suficiente contenido y comentarios con los que trabajar, finalmente Twitter fue descartada como opción, a pesar de que herramientas como el paquete `rtweet` nos permitían un uso inmediato de su API, obteniendo multitud de datos para cada comentario, como la geolocalización, el número de *likes* o información variada sobre el usuario.

En segundo lugar, al tiempo que estudiábamos la viabilidad de Twitter y de la BGG, intentábamos extraer información de Kickstarter, página de mecenazgo ya mencionada en varios puntos, por cuestiones semejantes, aunque esta página no disponía de una referencia para su API, semejante a la de la BGG pero devolviendo archivos de tipo JSON. Por otro lado, descubrimos más adelante que en [37] se había hecho ya el proceso de *web scrapping* al que le habíamos dedicado ya un tiempo, y aunque esta información se podía combinar con facilidad con los objetos *bgg_games* generados, nos encontramos con que encontrar parejas era más difícil de lo que parecía en un principio,

obteniendo un conjunto pequeño, con el que no demasiadas conclusiones podrían extraerse. Debido a eso, pero principalmente por la falta de tiempo y la excesiva extensión, acabamos descartando esta plataforma, quedándonos únicamente con los resultados obtenidos a partir de la BGG, como los que hemos visto en las secciones 4.2.2 y 6.4.

Otro punto que destacar es que habíamos pensado en recurrir a métodos de reducción de la dimensionalidad, como el escalamiento multidimensional (*MDS*) o al análisis de componentes principales (*PCA*), para obtener una jerarquía dentro de las variables, ordenas por relevancia. Además, mezclando estas técnicas con el análisis factorial, nos hubiese gustado ser capaces de construir una nueva variable que midiese la "popularidad" de un juego, agrupando tanto el movimiento dentro de una plataforma como el sentimiento asociado o incluso su pertenencia o no a ciertas categorías. Esta idea se dificultó considerablemente por el tamaño de los datos con los que estábamos trabajando, si bien comprobamos que podemos incluir dentro de una receta del paquete `recipes` el aplicar MDS, aunque es algo recomendado únicamente para valores numéricos, cuestión que resultaba limitante en nuestro caso.

Por último, si recordamos, en el capítulo 5 asumimos que todos los comentarios con los que trabajábamos se encontraban en inglés, cuestión que un primer momento queríamos estudiar. Para lograr este objetivo, estudiamos las funciones incluidas en el paquete `koRpus` [7], entre las que se incluía *guess_lang*, herramienta que nos permitía, dado un corpus de referencia, generalmente la Declaración de los Derechos Humanos en varios idiomas, identificar el idioma de un texto, ofreciendo además varias opciones, ordenadas por importancia. Sin embargo, aunque al principio nos pareció un buen añadido al trabajo, pues también incluía funciones de minería de texto como las del paquete `tm`, rápidamente vimos que era una herramienta mucho más compleja de lo que necesitábamos, y de hecho su principal utilidad, *guess_lang*, daba malos resultados para textos de longitud reducida, como la de los comentarios considerados. Por esas razones, finalmente optamos por recurrir a los paquetes `tidytext` y `tm`, más adecuados para nuestros objetivos.

7.3 Líneas futuras de trabajo

A pesar de haber obtenido multitud de resultados a lo largo del trabajo, es indudable que hay muchos de ellos que pueden mejorarse recurriendo a herramientas más potentes, y otros que simplemente no se han llegado a abordar, o que no nos hemos

extendido tanto en su estudio como nos gustaría, meramente por cuestiones de extensión y temporales.

El primer lugar, hablando del modelo vectorial en el procesamiento del lenguaje natural, ya vimos que teníamos varios defectos que lo limitaban considerablemente, pudiendo corregirse algunos de ellos con relativa facilidad. Comprobamos en multitud de casos la importancia de la elección del vocabulario, especialmente del específico del ámbito que hay que tratar independientemente, el que nosotros denominados *games_vocab*, pues puede condicionar tanto los resultados como las gráficas. Una forma de mejorar esta selección es hablar con un experto, para obtener información sobre el vocabulario específico del tema, sobre si hay algún conjunto de palabras que se usen de formas diferentes a las habituales, o cualquier otra cuestión que sea relevante a la hora de extraer conclusiones a partir del texto. Otro añadido sería el recurrir al *stemming*, reduciendo cada palabra a su raíz, si bien de esta forma podríamos estar perdiendo capacidad de interpretación.

Por otra parte, siguiendo con el modelo vectorial y con el análisis del sentimiento, una forma de mejorarlos es considerar, junto con los pesos o el sentimiento, la correlación de cada palabra del vocabulario con la anterior o siguiente palabra, para los digramas y trigramas más comunes. Así, podríamos ponderar diferentes apariciones de una misma palabra dependiendo del contexto en el que la encontremos, diferenciando, por ejemplo, "*version*" de "*best version*", "*better version*" o de las versiones negadas de dichas expresiones. Así tendríamos en cuenta también las partículas negativas, que podrían condicionar el significado de la palabra siguiente, hecho que nuestros modelos no tienen en cuenta. Para aplicar estos modelos, sin embargo, es necesario recurrir a motores de procesamiento mucho más potentes que los aquí estudiados.

Dada la importancia de los juegos de tipo *party* y de los *wargames,* así como de los juegos de estilo alemán, sería interesante repetir un análisis semejante al aquí realizado, pero únicamente teniendo en cuenta dichas categorías, para comprobar si las impresiones que hemos recibido sobre dichos tipos, mayoritariamente positivas, son correctas.

Centrándonos en el punto de la predicción de valoraciones, ya vimos que los modelos considerados proporcionaban resultados hasta cierto punto aceptables, obteniendo el mejor rendimiento con una red neuronal relativamente pequeña. Estos hechos combinados nos podrían hacer pensar que los datos con los que hemos trabajado tienen bastante ruido, y por lo tanto sería interesante seguir por este camino, recurriendo a redes neuronales más potentes, a costa de la interpretabilidad del modelo. De cualquier

forma, tanto el incorporar nueva información como dar explicación a aspectos derivados de los datos de partida originales nos debería facilitar el hacer una elección de un modelo más apropiado, mejorando de esta forma nuestra interpretación del mercado y nuestra capacidad predictiva. Una opción, ya planteada en varios puntos del proyecto, sería recurrir al escalamiento multidimensional o al análisis de componentes principales, aprovechando la correlación presente entre diversas variables, obteniendo así más conocimiento sobre los datos con los que trabajamos, lo que repercutirá con toda seguridad en una mejora de los resultados obtenidos.

Por otro lado, en un trabajo futuro podríamos explorar la opción de ver qué fragmentos de nuestro código podrían quedar más elegantes usando funciones de orden superior, como las funciones de tipo *map* del paquete `purrr`, si bien cuando empecé a aprender sobre estas funcionalidades una gran parte del código ya estaba desarrollada, y la cantidad de frentes abiertos no permitió que esta refactorización alcanzara la prioridad necesaria, siguiendo por ello nuestra alternativa el uso a lo largo del proyecto de la programación imperativa, recurriendo a bucles y a condicionales en su lugar.

En último lugar, una mejor representación de determinados resultados, tales como las nubes de palabras por categorías, así como la construcción de funciones que las construyesen, actualizándose automáticamente al introducirse en la base de datos un comentario, podrían ser utilidades relevantes para páginas como la BGG o Kickstarter, y pudiéndose extender estas a cualquier otra página que busque representar grandes conjuntos de opiniones con facilidad.

Sin embargo, a pesar de habernos encontrado con multitud de dificultades y de habernos vistos obligados a descartar varios caminos, consideramos que este proyecto puede considerarse como un primer paso prometedor hacia el análisis en profundidad del mercado de los juegos de mesa, ofreciéndose en él resultados concretos y generales, así como ideas futuras que podrían enriquecer el trabajo aquí expuesto.

Bibliografía

[1] Cooksey, B. (2014). *An Introduction to APIs*. Zapier, Inc.

[2] Feinerer, I. (2021). *Introduction to the tm Package*. Disponible en: https://cran.r-project.org/web/packages/tm/vignettes/tm.pdf.

[3] Kuhn, M. y Silge, J (2021). *Tidy Modeling with R*.

[4] Kwartler, T (2017). *Text Mining in Practice with R.* Chichester, West Sussex: Wiley.

[5] Lui, B. (2021). *Sentiment Lexicon*. Disponible en: https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html.

[6] Martín, F. J. y Ruiz, J. L (2016). *Procesamiento del Lenguaje Natural*. Disponible en: https://www.cs.us.es/cursos/aia-2016/temas/tema-04.pdf

[7] Michalke, M., Brown, E., Mirisola, A., Brulet A. y Hauser, L. (2021). *Package 'koRpus'*. Disponible en: https://cran.r-project.org/web/packages/koRpus/koRpus.pdf.

[8] Mohammad, S. y Turnet, P. (2021). *NRC Word-Emotion Association Lexicon*. Disponible en: http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm.

[9] Nielsen, F. (2021). *Sentiment Lexicon*. Disponible en: http://www2.imm.dtu.dk/pubdb/pubs/6010-full.html.

[10] *Package 'dplyr'* (2021). Disponible en: https://cran.r-project.org/web/packages/dplyr/dplyr.pdf.

[11] *Package 'ggplot2'* (2021). Disponible en: https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf.

[12] *Package 'tidymodels'* (2021). Disponible en: https://www.tidymodels.org/.

[13] *Package 'tidyr'* (2021). Disponible en: https://cran.r-project.org/web/packages/tidyr/tidyr.pdf.

[14] *Package 'tidyverse'*, (2021). Disponible en: https://www.tidyverse.org/.

[15] *Package 'tm'* (2021). Disponible en: https://cran.r-project.org/web/packages/tm/tm.pdf.

[16] *Package 'wordcloud'* (2021). Disponible en: https://cran.r-project.org/web/packages/wordcloud/wordcloud.pdf.

[17] *Package 'yardstick'* (2021). Disponible en:

https://cran.r-project.org/web/packages/yardstick/yardstick.pdf.

[18] Silge, J. y Robinson, D. (2017). *Text Mining with R: A Tidy Approach*. O'Reilly Media, Inc.

[19] Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer.

[20] Wickham, H. y Grolemund, G. (2017). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data.* Sebastopol, California: O'Reilly Media, Inc.

[21] Wilkinson, L. (2005) *The Grammar of Graphics (Statistics and Computing)*. Chicago, IL: Springer.

[22] *BoardGameGeek FAQ* (2021). Disponible en:

https://boardgamegeek.com/wiki/page/BoardGameGeek_FAQ.

[23] *BoardGameGeek Data Mining* (2021). Disponible en:

https://boardgamegeek.com/wiki/page/Data_Mining#.

[24] *BGG XML API* (2021). Disponible en:

https://boardgamegeek.com/wiki/page/BGG_XML_API#.

[25] *Board Game Arena* (2021). Disponible en: https://es.boardgamearena.com/.

[26] *Devir, Catán* (2021). Disponible en: https://devir.es/producto/catan/.

[27] E*scharts, Games* (2021). Disponible en: https://escharts.com/games.

[28] *Grand View Search, Playing Cards And Board Games Market Size, Share & Trends Analysis Report By Product (Board Games (Chess, Scrabble, Monopoly, Ludo), Playing Cards), By Distribution Channel, And Segment Forecasts, 2019 - 2025* (2019). Disponible en: https://www.grandviewresearch.com/industry-analysis/playing-cards-board-games-market.

[29] *Wikipedia - Juegos de estilo alemán* (2021). Disponible en:

https://es.wikipedia.org/wiki/Juego_de_estilo_alem%C3%A1n.

[30] *Matrix: Simple Triplet Matrix* (2021). Disponible en:

https://rdrr.io/cran/slam/man/matrix.html.

[31] *SimilarWeb* (2021). Disponible en:

https://www.similarweb.com/top-websites/category/games/board-and-card-games/.

[32] *SimilarWeb, Universidad de Sevilla* (2021). Disponible en:

https://www.similarweb.com/website/us.es/.

[33] *SimilarWeb, BoardGameGeek* (2021). Disponible en:

https://www.similarweb.com/website/boardgamegeek.com/#overview.

[34] *SimilarWeb, Kickstarter* (2021). Disponible en: https://www.similarweb.com/top-websites/category/games/games/.

[35] *TIOBE Index for June 2021* (2021). Disponible en: https://www.tiobe.com/tiobe-index/.

[36] *TwitchTracker, Chess* (2021). Disponible en: https://twitchtracker.com/games/743.

[37] *Webrobots, Kickstarter Datasets* (2021). Disponible en:

https://webrobots.io/kickstarter-datasets/