Analizador Visual de Corpus Lingüísticos

Trabajo de Fin de Grado

Ingeniería Informática

Septiembre de 2024

David Prieto Santos

_____

Roberto Therón Sánchez

_____

VNiVERSiDAD D SALAMANCA
CAMPUS DE EXCELENCIA INTERNACIONAL

Facvltad D Ciencias
VNiVERSiDAD D SALAMANCA

Certificado del/los tutor/es TFG

D. Roberto Therón Sánchez, profesor/a del Departamento de Informática y Automática de la Universidad deSalamanca,

HACE/N CONSTAR:

Que el trabajo titulado "Analizador Visual de Corpus Lingüísticos que se presenta, ha sido realizado por David Prieto Santos, con DNI ****7461H y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado en Ingeniería Informática en esta Universidad.

Salamanca, 1 de septiembre de 2024

THERON SANCHEZ ROBERTO - 07976246F

Digitally signed by THERON SANCHEZ ROBERTO - 07976246F
Date: 2024.09.04 14:13:47 +02'00'

Fdo.: _____

Resumen

En las ciencias cuantitativas el trabajo empírico ha evolucionado de forma que la visualización de datos se ha convertido en una parte esencial del mismo. De hecho, en muchos casos resulta difícil realizar un análisis o comunicación de datos sin la ayuda de herramientas gráficas. Esto cobra una vital importancia a la hora de analizar un Corpus lingüístico (conjunto, normalmente muy amplio, de ejemplos reales de uso de una lengua), ya que los datos extraídos del mismo tienen una estructura compleja y cuentan con múltiples características de potencial interés.

Con este proyecto se pretende, a través de técnicas de Analítica visual, recolectar datos, en principio abstractos, para que por medio de representaciones gráficas y la interacción con ellas se proporcione conocimiento al ser humano. Los usuarios podrán acceder a la plataforma desde un navegador web y utilizar su propio corpus como entrada (también se ofrecen algunos corpora de prueba en la plataforma).

De este modo, los usuarios podrán analizar sus corpora visualmente a través de gráficos pensados específicamente para ello, obteniendo representaciones únicas y, en muchas ocasiones, interactivas.

El desarrollo de esta plataforma se ha llevado a cabo empleando los lenguajes de programación Markdown, HTML y Javascript a través de Observable Framework, un generador de sitios estáticos de código abierto para aplicaciones de datos, paneles, informes, etc.

Palabras clave: Corpus Lingüístico, Analítica visual, representaciones gráficas, navegador web, HTML, Javascript, Observable Framework.

Abstract

In the quantitative sciences, empirical work has evolved in such a way that data visualization has become an essential part of it. In fact, in many cases, it is difficult to perform data analysis or communication without the help of graphical tools. This is of vital importance when analyzing corpus linguistics (a typically large set of real-life examples of language use), as the data extracted from it have a complex structure and contain multiple features of potential interest.

This project aims to, through visual analytics techniques, collect initially abstract data so that, through graphical representations and interaction with them, knowledge is provided to humans. Users will be able to access the platform from a web browser and use their own corpus as input (some test corpora are also provided on the platform).
In this way, users will be able to visually analyze their corpora through graphs specifically designed for this purpose, obtaining unique and, in many cases, interactive representations.

The development of this platform has been carried out using the Markdown, HTML and JavaScript programming languages through Observable Framework, an open-source static site generator for data applications, dashboards, reports, etc.

Keywords: Linguistic Corpus, Visual Analytics, Graphical Representations, Web Browser, HTML, JavaScript, Observable Framework.

Agradecimientos

Me gustaría agradecer en primer lugar a mi tutor de TFG Roberto por haberme acompañado y guiado en este viaje.

Me gustaría dar las gracias también al resto de profesores que he tenido a lo largo de esta carrera en la USAL, que me han dado los conocimientos y la actitud necesarios para realizar un proyecto como este.

Por último, agradecer a todos los compañeros de la carrera, que han sido parte de todo este viaje, a mi familia y amigos, y todas las personas que me han apoyado y que me han llevado hasta aquí.

Tabla de figuras

1. Introducción

La lingüística es una disciplina altamente cuantitativa (y aún más cuando se tratan corpora).

Los datos extraídos de un corpus a menudo tienen una estructura compleja y vienen con múltiples características de interés potencial. Esto se debe en parte al creciente número y tamaño de corpora ricamente anotada y estratificada. Además, la automatización de extracción y anotación de datos da lugar a características adicionales para el análisis a un bajo coste.

Como resultado, actualmente los corpora lingüísticos a menudo terminan con un conjunto de observaciones con múltiples capas de características. Por ejemplo, los textos extraídos de un corpus hablado pueden haber sido producidos en un determinado entorno comunicativo, por un orador con ciertas características sociales, y, dependiendo de la estructura de interés, pueden venir con varios factores internos a considerar. Del mismo modo, las muestras de un corpus escrito pueden ser rastreadas a un texto escrito por un autor en particular, que data de un cierto período de tiempo, y que representa un género específico.

Todos estos factores se combinan de nuevo con parámetros internos del lenguaje. Dado el carácter inherentemente cuantitativo, así como la naturaleza potencialmente multifacética de los corpora, puede haber pocas dudas sobre la pertinencia de las herramientas visuales para esta rama de la lingüística.

Con este contexto se origina la idea principal de este TFG: desarrollar una herramienta que permita la creación de representaciones visuales específica para corpora lingüísticos. Primando la usabilidad y accesibilidad de las herramientas desarrolladas, así como la interactividad del usuario con las representaciones. Todo ello para ofrecer una respuesta con una mayor riqueza informativa que la mostrada en representaciones gráficas convencionales.

2. Objetivos del proyecto

Los objetivos del proyecto que se proponen son:

- Desarrollar y/o adaptar herramientas visuales interactivas para el análisis de grandes corpora lingüísticos.

- Primar la usabilidad y accesibilidad de las herramientas desarrolladas, así como la facilidad de ampliación posterior de la herramienta.

- Las herramientas desarrolladas deberán funcionar en un navegador Web.

3. Técnicas y herramientas

3.1 Herramientas utilizadas

En este apartado se explicarán las herramientas utilizadas finalmente para el proyecto.

- GitHub [1]: plataforma en línea que permite a desarrolladores y equipos de trabajo gestionar, almacenar y colaborar en proyectos de software utilizando Git, un sistema de control de versiones distribuido. Permite llevar un control de las versiones que se han implementado a lo largo del desarrollo.

- Visual Studio Code [2]: editor de código fuente ligero pero potente desarrollado por Microsoft. Combina ligereza y funcionalidad, siendo una herramienta muy apreciada tanto por desarrolladores principiantes como por profesionales experimentados.

- Observable Framework [3]: conjunto de herramientas y componentes de interfaz de usuario que facilita la creación de aplicaciones interactivas basadas en datos. Esta plataforma está diseñada para hacer que la creación de visualizaciones, análisis de datos y aplicaciones interactivas sea más sencilla y accesible.

- Markdown [4]: lenguaje de marcado ligero y fácil de usar que permite crear contenido estructurado utilizando texto plano. Fue creado por John Gruber en 2004 con la intención de ser una forma simple y legible de formatear texto para la web, que pueda ser convertido a HTML de manera sencilla.

  Se ha vuelto muy popular, especialmente en comunidades de desarrolladores, debido a su facilidad de uso y su capacidad para integrarse con diversas herramientas y plataformas.

- HTML [5]: es el lenguaje de marcado estándar utilizado para crear y estructurar contenido en la web. Fue desarrollado por Tim Berners-Lee en 1991, y desde entonces ha evolucionado a través de varias versiones, con HTML5 siendo la versión más reciente y ampliamente utilizada.

  HTML es esencial para la creación de la web, ya que define cómo se organiza y presenta el contenido, sirviendo como el pilar fundamental sobre el cual se construyen todas las páginas web.

- Javascript [6]: lenguaje de programación interpretado y de alto nivel que se utiliza principalmente para crear contenido dinámico e interactivo en las páginas web. Fue creado por Brendan Eich en 1995 y es uno de los lenguajes fundamentales en el desarrollo web, junto con HTML y CSS.

JavaScript es esencial en el desarrollo web moderno, proporcionando las herramientas necesarias para crear experiencias de usuario ricas e interactivas. Su versatilidad y amplio ecosistema lo han convertido en uno de los lenguajes de programación más populares y demandados en la industria.

- Node.js [7]: un entorno de ejecución de JavaScript que permite ejecutar código JavaScript del lado del servidor, fuera del navegador. Ha revolucionado el desarrollo backend al permitir que los desarrolladores utilicen JavaScript en el servidor, simplificando la construcción de aplicaciones rápidas, escalables y eficientes.

- NPM [8]: es el gestor de paquetes predeterminado para Node.js, utilizado para instalar, compartir y gestionar librerías y herramientas en proyectos de desarrollo JavaScript. Proporciona un ecosistema robusto y eficiente para la gestión de dependencias y la automatización de tareas en proyectos de software.

- D3.js [9]: biblioteca de JavaScript que permite crear visualizaciones de datos dinámicas e interactivas en el navegador utilizando estándares web como SVG (Scalable Vector Graphics), HTML5 y CSS.

3.2 Técnicas utilizadas

Metodología SCRUM

Para el proyecto se usó la metodología SCRUM. Scrum es una metodología ágil utilizada en la gestión y desarrollo de proyectos, especialmente en el campo del desarrollo de software. Su enfoque se basa en la entrega incremental y continua de productos funcionales, permitiendo a los equipos responder rápidamente a los cambios y mejorar la eficiencia y la calidad del trabajo.

Esta metodología es la idónea para un trabajo de fin de grado ya que puede haber cambios continuos y nos brinda una gran flexibilidad.

Además, el trabajo se organiza en sprints (ciclos de trabajo de 1 a 4 semanas en los que el equipo trabaja en una porción del producto, que debería ser funcional al final de este), que se adaptan perfectamente al calendario de reuniones con el tutor (una cada semana o cada dos semanas).

En estas reuniones se presenta el resultado del sprint para identificar lo que funcionó bien, lo que no y cómo se puede mejorar en el próximo sprint.

De esta forma se puede abordar un proyecto que al principio puede parecer complejo de manera eficiente y adaptativa.

4. Aspectos relevantes sobre el desarrollo del proyecto

En este apartado se exponen los aspectos relevantes del proyecto, donde se analizarán y discutirán las diferentes partes que componen la aplicación en cuestión.

4.1 Proyecto Inicial

En este apartado se va a exponer el proyecto pensado en su estado inicial.

La idea inicial consistía en desarrollar una herramienta que, a partir de un corpus lingüístico de gran tamaño, ofreciera al usuario distintas representaciones de datos extraídos de dicho corpus, buscando siempre que fuera posible la interactividad con estas representaciones.

La página mostrada al usuario incluirá un buscador que, una vez el usuario ha especificado una palabra en él, se le muestran distintos gráficos que muestran de forma rápida y eficaz las características de dicha palabra, así como las relaciones de esta palabra con el resto. Similar al funcionamiento de la página del CORPES [10] (Corpus del Español del Siglo XXI).

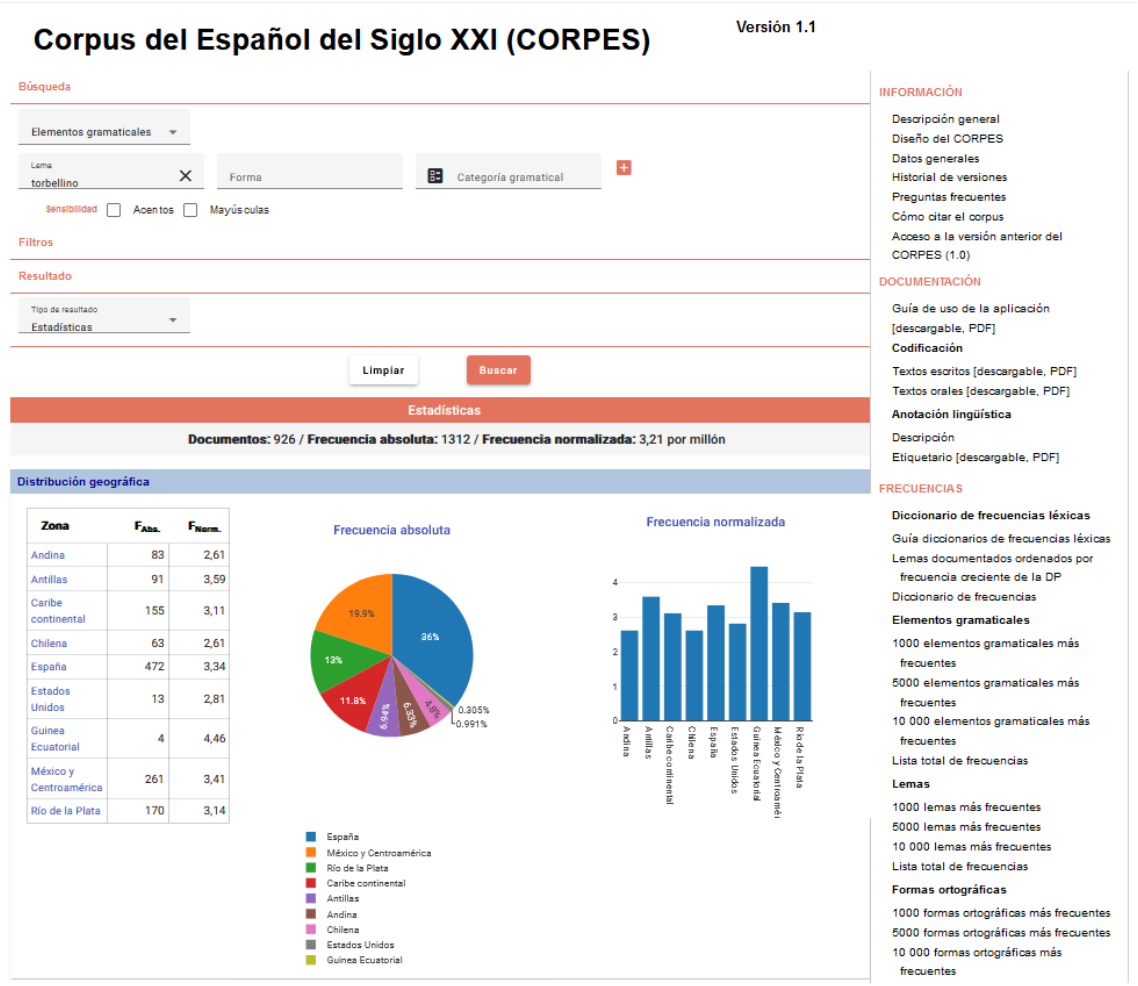

Figura 1: Página Web del CORPES

Como se puede ver en la Figura 1, en esta página se presentan unos datos generales y unos gráficos básicos para representarlos. Nuestra idea era hacer algo similar, aunque fomentando la interactividad y mostrando información más rica.

Antes de empezar a desarrollar la propia aplicación, estuve trabajando un tiempo con la herramienta Vega-Lite [11], que es una gramática de alto nivel para la creación de gráficos interactivos. Esta herramienta cuenta con una sintaxis JSON declarativa para crear un gran rango de visualizaciones para el análisis de datos.

Además, en la página de Vega-Lite, disponible en el apartado de Bibliografía, se proporciona una extensa documentación, así como un editor en el que realicé mis primeros gráficos online.

Después de un tiempo trabajando con el editor, me pareció increíble que con una sintaxis tan concisa se pudieran realizar unas representaciones que dieran tan buen resultado.

Figura 2: Gráfico de barras con Vega-Lite

Como se puede ver en la Figura 2, se puede representar un gráfico de barras con tan solo unas pocas líneas de código, teniendo en cuenta que incluso los datos que se representan se incluyen en dicho código.

Tras realizar algunas representaciones sencillas, busqué una base de datos en Kaggle [22] para empezar a trabajar con un conjunto de datos más real. Encontré una buena base de datos en español que estaba compuesta de artículos de Wikipedia.

Sin embargo, mi base de datos ahora estaba compuesta por texto plano, por lo que desarrollé unos programas sencillos para “tokenizar” la base de datos y llevar la cuenta de las veces que aparecía cada palabra.

Con este nuevo corpus creé algunos gráficos básicos, como gráficos de barras o diagramas de sectores (con estos últimos me di cuenta de que Vega-Lite también creaba una leyenda para identificar mejor los datos automáticamente, otra característica muy positiva) y me sentí preparado para empezar a trabajar en la propia herramienta.

4.2 Proyecto en Observable Framework

Ya que la aplicación debe ser accesible por los usuarios a través de un navegador web, mi tutor me planteó el uso de Observable Framework, que permite la creación de páginas web utilizando Markdown, HTML y Javascript. Además, y como ya se ha mencionado anteriormente, en el apartado de Herramientas utilizadas, se centra en la representación visual y el análisis de datos, por lo que es perfecto para este proyecto.

Este entorno es a la vez:

- Un servidor de desarrollo local, muy útil para previsualizar el proyecto localmente mientras se está desarrollando. La página se actualiza instantáneamente con los cambios.
- Un generador de sitios que compila Markdown y Javascript (entre otros) junto a instantáneas de datos (locales o no) generadas por cargadores.
- Una interfaz de línea de comandos para automatizar el despliegue de forma que se pueda compartir la página rápidamente y de forma segura.

A la hora de crear el proyecto se puede realizar de forma muy sencilla con la ayuda de Node.js y NPM, aunque no voy a entrar en detalles (en la propia página del Framework se incluye un tutorial que te guía paso a paso para hacerlo).

Resulta igual de sencillo desplegar el proyecto en web y la inclusión de librerías de Javascript gracias a NPM, aunque el Framework ya incluye por defecto varias librerías relacionadas con la representación de datos (mencionaré varias opciones más adelante).

Como ya he dicho, siempre hemos querido darle mucha importancia a la interactividad del usuario con la herramienta y este Framework ofrece una gran ventaja para conseguirla, ya que se ejecuta como una hoja de cálculo: el código se ejecuta automáticamente cuando cambian las variables referenciadas.

De esta forma, se consigue una fácil interactividad ya que es el propio Framework el que mantiene el estado en sincronía. También facilita la programación asincrónica gracias a la espera implícita de promesas y se consigue un gran rendimiento y mayor flexibilidad a la hora de escribir el código.

Además de todo esto, el Framework también proporciona herramientas como generadores (entradas interactivas, parámetros de animación) o inputs (elementos gráficos de interfaz como desplegables, cajas de texto, etc.) que favorecen enormemente la interactividad.

Una vez visto este breve resumen del Framework, se puede entender por qué se acabó usando para el desarrollo de la aplicación, y es que encaja perfectamente con esta.

Tras hacer un framework de prueba siguiendo el ejemplo que proporciona Observable y probar distintas opciones para familiarizarme con la herramienta, empecé a integrar gráficos de Vega-Lite a un framework vacío.

Una vez representados los gráficos básicos, era hora de empezar a darle forma a la aplicación como queríamos por lo que intenté añadir un buscador de palabra. Para ello, y buscando información en Vega-Lite, encontré la posibilidad de añadir filtros a los propios gráficos.

Parecía una buena opción, ya que además la proporcionaba la propia Vega-Lite, sin embargo, este filtro se podía aplicar solo al gráfico al que pertenecía, y la idea era representar varias estadísticas con el mismo buscador.

Buscando más a fondo encontré la posibilidad de introducir parámetros en los gráficos. De esta forma y con un input externo, como los que ofrece Observable, se pueden enlazar varias representaciones a un mismo filtro. El problema fue en este caso que Observable utiliza la librería de javascript de Vega-Lite, por lo que la sintaxis cambia ligeramente. Hasta ahora no había sido un problema, porque los cambios eran mínimos, pero los parámetros deben de cambiar en mayor medida.

Por ello, y tras un tiempo probando distintas opciones y buscando ejemplos, casi la mayoría de ellos con la sintaxis de JSON, pensé que lo mejor sería probar otra librería de representación de datos de las que incluye Observable.

4.3 Librerías de visualización

Como ya he adelantado, Framework ofrece por defecto varias librerías para la visualización de datos. En este apartado vamos a ver las distintas opciones que se contemplaron con sus ventajas e inconvenientes.

Vega-Lite

Ya se ha visto el uso de esta librería hasta ahora, pero quería utilizar este apartado a modo de resumen, y para concretar ventajas e inconvenientes y exponer unas últimas conclusiones.

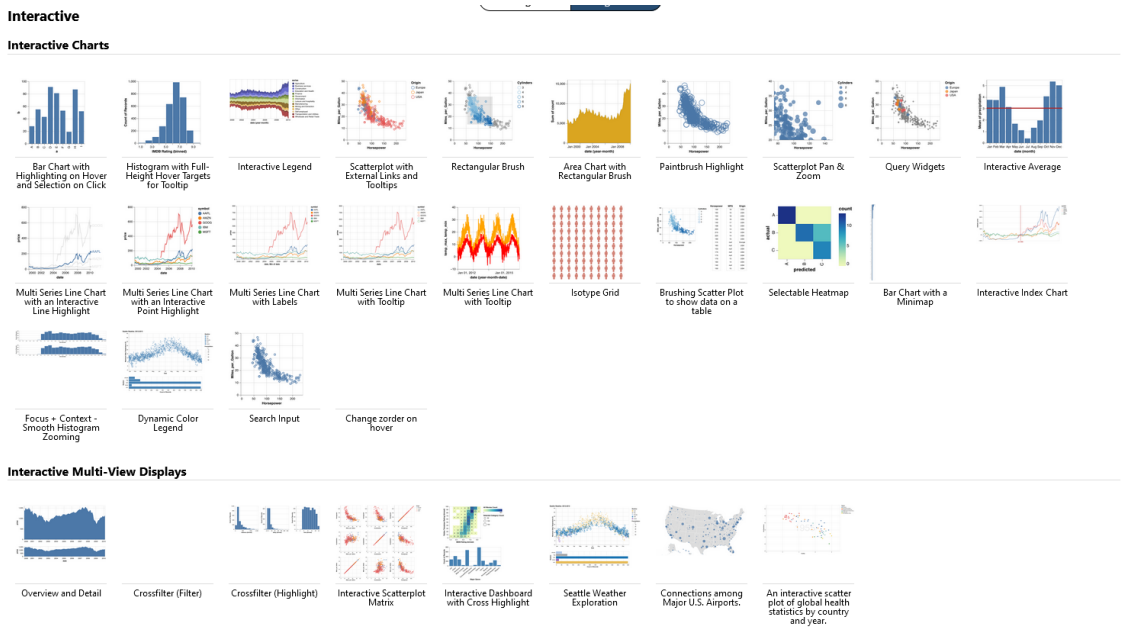

Figura 3: Ejemplos de gráficos con Vega-Lite

En la Figura 3 se pueden ver algunos de los ejemplos de gráficos que proporciona Vega-Lite, interactivos en este caso. Como ya se mencionó anteriormente, ofrece mucha variedad, además de interactividad y facilidad de uso.

Sin embargo, Observable usa su librería de javascript. Esta librería no cuenta con tanta documentación y ejemplos como su formato a JSON y, al cambiar la sintaxis, llegó un punto en el que me vi obligado a cambiar de librería.

Esto podría ser considerado un defecto de Observable y no de Vega-Lite, pero aún así parecía adecuado revisar el resto de las opciones que ofrecía Observable en vez de obsesionarse con Vega-Lite.

Observable Plot

Revisando el resto de las librerías que incluía el Framework encontré Observable Plot [12], y pensé que sería buena opción, ya que está desarrollada por la propia Observable.

Al ojear su página web, disponible en el apartado de Bibliografía, vi que ofrecía una buena variedad de representaciones, y visitando una galería de gráficos que proporciona Observable reparé en que permitía crear gráficos de forma aún más concisa que Vega-Lite, pudiendo hacer representaciones incluso con una sola línea de código, sin dejar de obtener muy buenos resultados.

Figura 4: Ejemplos de gráficos con Observable Plot

Como se puede ver en la Figura 4, en esta galería se exponen todavía un mayor número de gráficos (solo se muestran los gráficos relacionados con barras o celdas propuestos por Observable, hay muchas más opciones).

Al ver estas representaciones pensé que podía ser una buena librería y, al ser de una sintaxis tan sencilla, facilita mucho el aprendizaje. Empecé por realizar unos gráficos similares a los realizados anteriormente con Vega-Lite, esta vez pudiendo incorporar un filtro de palabra conectado a varias representaciones.

Con el filtro de palabra funcionando, era la hora de representar algo más que simplemente el número de apariciones de dicha palabra. Buscando en páginas de

búsqueda en corpus, como la de CORPES ya mencionada, me di cuenta de que una estadística común a representar son los n-gramas, conjuntos de n palabras, o colocados, palabras que se suelen usar cerca de otra. Me pareció una característica interesante a analizar para la búsqueda de una palabra en un corpus así que decidí incorporarlo.

Para ello, desarrollé un pequeño programa que dividía los textos en n-gramas de 2 palabras (o bigrama). De esta forma obtuve dos gráficos nuevos: uno que mostraba las palabras que precedían a la buscada más habitualmente y otro para las palabras que la seguían.

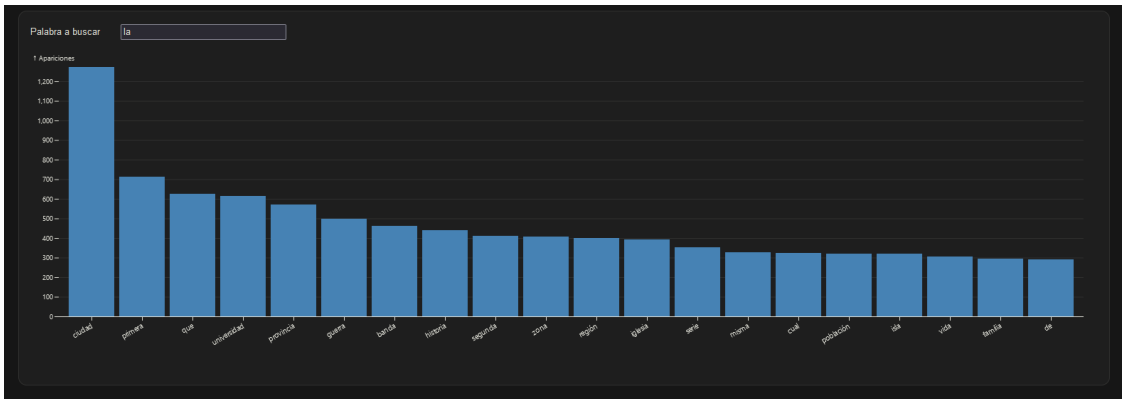

Figura 5: Gráfico de n-gramas con Observable Plot

Obteniendo los resultados esperados (Figura 5), ya que hasta ahora no había representado más que gráficos de barras, indagué un poco más y observé que, aunque cuenta con una sintaxis concisa, Plot permite customizar los gráficos y ofrece algo más de lo que parece a simple vista.

```
<!-- Filtro -->

```js
const palabraInput = Inputs.text({label: "Palabra a buscar"});
const palabra = Generators.input(palabraInput);
```

<!-- Apariciones de cada palabra -->

<div class="card" style="display: flex; flex-direction: column; gap: 1rem;">
  ${palabraInput}
  ${resize((width) => Plot.plot({
    width,
    marginBottom: 80,
    y: {grid: true, label: "Apariciones"},
    x: {label: null, tickRotate: -30},
    marks: [
      Plot.barY(bigramas.filter((d) => d.word === palabra),
        Plot.groupX(
          {
            y: "count"
          },
          {
            x: "bigram",
            sort: { x: "y", reverse: true, limit: 20 },
            fill: "steelblue"
          }
)
      ),
      Plot.ruleY([0])
    ]
  }))}
</div>
```

Figura 6: Código correspondiente a la Figura 5

Antes de continuar, quiero repasar el código correspondiente a la Figura 5, mostrado en la Figura 6.

Primero se define el filtro de Observable Inputs, con una sintaxis muy sencilla y a continuación, se muestran tanto el filtro como el gráfico en una división de HMTL.

La construcción del gráfico consta de varias partes: primero se definen las dimensiones, en este caso el ancho de la ventana y la altura que necesite, dejando un margen en la parte baja para el texto; luego se definen los ejes Y, especificando el uso de cuadrícula y la etiqueta con el que se mostrará, y X, en este caso sin usar etiqueta y rotando el texto, para que no se solapen las palabras largas; a continuación se definen los datos con los que se construirán las barras, filtrando el corpus para aislar aquellas que coincidan con la palabra buscada; luego se definen los datos correspondientes a cada eje: en el X, el campo “bigram”, correspondiente a la segunda palabra del bigrama, ordenados según el eje Y de mayor a menor y con un máximo de 20 elementos a mostrar, y en el Y la cuenta para ese bigrama; por último, se resalta el eje X.

Continuando con la evolución de la aplicación, no encontré muchas más características que mostrar de mi corpus, ya que era simple texto plano, por lo que empecé a buscar uno nuevo en Kaggle. Tras un tiempo sin encontrar un corpus que me convenciera, decidí añadir más características en los programas de contabilización del corpus. Implementé una parte que añadía para cada palabra una fecha, de 1990 a 2024 y un lugar, de habla hispana, al azar.

Con estos nuevos datos probé a representar otros tipos de gráficos y creé un diagrama de sectores tanto para la fecha como para el país. Aún así, seguían siendo gráficos muy sencillos, así que probé a representar un mapa de calor, mostrado en la Figura 7.

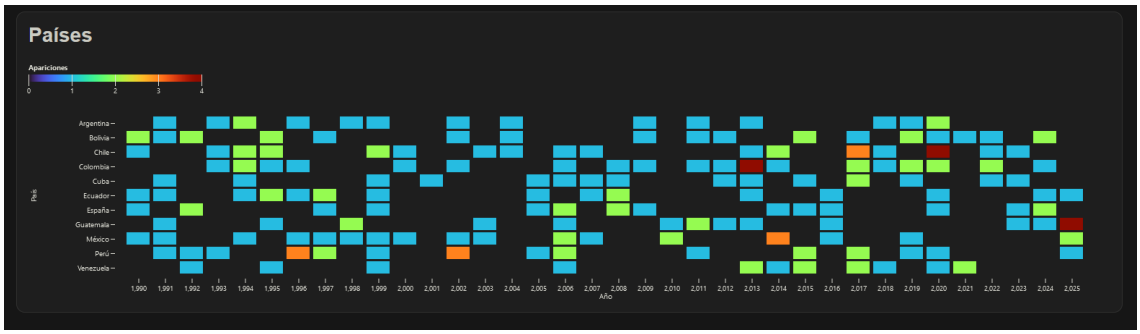

Figura 7: Mapa de calor con Observable Plot

Con estos gráficos y, tras discutirlo con el tutor, decidimos ampliar la búsqueda a corpora de otros idiomas, ya que la idea era disponer de datos reales y hasta ahora solo estaba trabajando con un conjunto del corpus, pero al trabajar con la totalidad del corpus y añadir características de forma aleatoria, los gráficos no mostrarían estadísticas con sentido.

Estuve examinando la página de English-Corpora [14] con mi tutor, que cuenta con varios copora en inglés y además ofrece fragmentos de ellos de manera gratuita, por lo que decidimos probarla. Realicé algunas representaciones y, aunque contaba con un mayor número de datos y estadísticas, pronto me encontré con un nuevo problema: Observable Plot apenas ofrece interacción con el usuario en sus gráficos, tan solo interacción relacionada con mostrar información adicional. Por ejemplo, mostrar los datos del elemento correspondiente a una barra al hacer clic en un diagrama de barras.

La interacción siempre ha sido un objetivo importante para nosotros y algo para tener en cuenta, fue por eso por lo que decidí probar otra librería que había visto por encima anteriormente.

D3

Mientras barajaba la opción de Plot vi algunos gráficos con D3 y pensé que eran unas buenas representaciones, pero con un enfoque de mucho más bajo nivel, por lo que al principio no la tuve muy en cuenta.

Sin embargo, tras revelarse la poca interactividad que ofrecía Plot, decidí darle una oportunidad e indagar un poco más

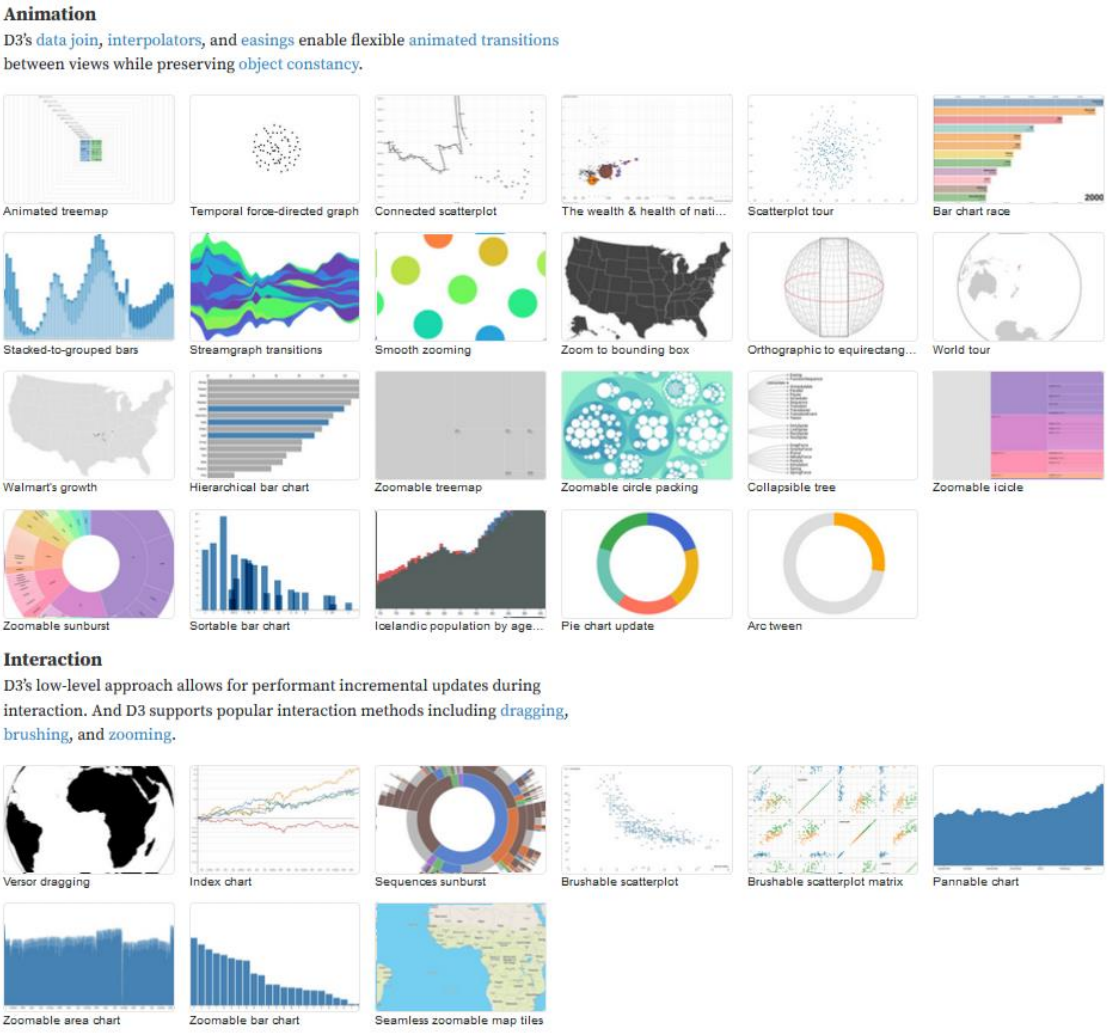

Figura 8: Ejemplos de gráficos con D3

En la Figura 8, muestro algunos de los gráficos que se pueden realizar con D3, en este caso son los gráficos con interactividad que propone Observable en su galería de D3. Como se puede ver, ofrece también una gran variedad de gráficos, muchos de ellos muy interesantes y, sobre todo, interactivos.

Siguiendo varios ejemplos y consultando continuamente la documentación (extensa y con mucha información, resuelve muchas dudas) pude crear mis primeras representaciones con D3 y terminar con una página similar a la anteriormente creada con Plot.

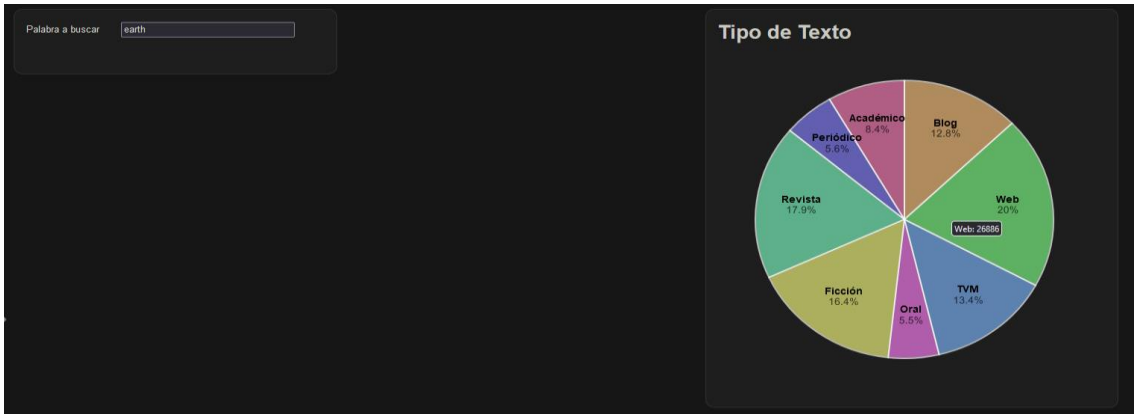

Figura 9: Página inicial con D3 (1 de 2)

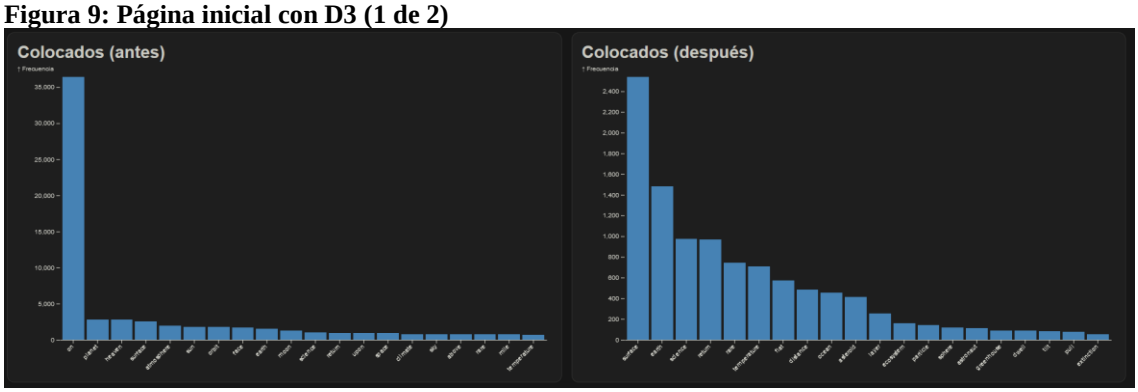

Figura 10: Página inicial con D3 (2 de 2)

Como se puede ver en las Figuras 9 y 10, los gráficos son similares a los creados anteriormente, aunque para el caso del diagrama de sectores añadí un "tooltip" al pasar el cursor por encima de los sectores.

Antes de seguir, quiero mostrar con algunas figuras la estructura que sigue a la hora de crear un gráfico con D3, ya que, aunque hay ciertas variaciones dependiendo del tipo, suele ser la misma o muy parecida. Para ello veremos el código correspondiente al primer gráfico de barras de la Figura 10.

```
133  ```js
134  // Dimensiones
135  var c_width = 600,
136      c_height = 500,
137      c_marginTop = 20,
138      c_marginRight = 0,
139      c_marginBottom = 50,
140      c_marginLeft = 70;
141
142  // Datos
143  var c_data = coll.filter((d) => d.lemma === palabra);
144  c_data = c_data.filter(function(d,i){
145    return i < 10;
146  });
```

Figura 11: Código correspondiente a estructura de gráfico inicial con D3 (1 de 5)

En la Figura 11 se muestra la primera parte del código de un gráfico, correspondiente a la definición de las dimensiones que tendrá este y al filtrado de datos.

```
148   // Escala X
149   var c_x = d3.scaleBand()
150     .domain(d3.groupSort(c_data, ([d]) => -d.freq, (d) => d.coll))
151     .range([c_marginLeft, c_width - c_marginRight])
152     .padding(0.1);
153
154   // Escala Y
155   var c_y = d3.scaleLinear()
156     .domain([0, d3.max(c_data, function(d) { return +d.freq; })])
157     .range([c_height - c_marginBottom, c_marginTop]);
158
```

Figura 12: Código correspondiente a estructura de gráfico inicial con D3 (2 de 5)

La segunda parte, como se puede ver en la Figura 12 se corresponde a la creación de escalas. Para ello, hay que especificar el dominio y el rango. En este caso, al tratarse de un diagrama de barras se necesita una escala lineal para el eje Y, con un dominio y rango discreto, correspondiente a la altura de la barra; y una escala de bandas para el eje X, similar a la del eje Y excepto que su rango de salida es continuo, especificando el ancho de la barra (inicio y final).

```
159   // SVG
160   const c_svg = d3.create('svg')
161     .attr('width', c_width)
162     .attr('height', c_height)
163     .attr('viewbox', [0, 0, c_width, c_height]);
164
```

Figura 13: Código correspondiente a estructura de gráfico inicial con D3 (3 de 5)

La tercera parte consiste en la creación del SVG. Un SVG es un gráfico vectorial escalable, escrito en lenguaje de marcado extensible XML y cuya especificación es un estándar abierto. Es muy sencillo adjuntarlo a una división de HTML, lo que nos viene perfecto, ya que la página de la aplicación está escrita en este lenguaje.

Este SVG contendrá toda la información del gráfico: dimensiones, la forma en que se dibuja, posiciones, colores y texto, así como su fuente, tamaño, etc.

En la Figura 13 se puede ver como se crea el SVG junto con unos atributos correspondientes a dimensiones del gráfico.

```
// Barras
c_svg.append('g')
  .attr('fill', 'steelblue')
  .selectAll()
  .data(c_data)
  .join('rect')
    .attr('x', (d) => c_x(d.coll))
    .attr('y', (d) => c_y(d.freq))
    .attr('height', (d) => (c_height - c_marginBottom) - c_y(d.freq))
    .attr('width', c_x.bandwidth());

```

Figura 14: Código correspondiente a estructura de gráfico inicial con D3 (4 de 5)

En la Figura 14 podemos ver la cuarta parte de creación de un gráfico, correspondiente a las divisiones, ya sean barras, sectores, etc., dependiendo del gráfico. Para las dimensiones de estas divisiones se usarán los parámetros correspondientes a cada elemento calculados antes y recogidos en las escalas.

```
// Eje X
c_svg.append('g')
  .attr('transform', `translate(0,${c_height - c_marginBottom})`)
  .call(d3.axisBottom(c_x).tickSizeOuter(0))
  .selectAll('text')
    .attr('transform', 'translate(-10,0)rotate(-45)')
    .style('text-anchor', 'end');

// Eje Y
c_svg.append('g')
  .attr('transform', `translate(${c_marginLeft},0)`)
  .call(d3.axisLeft(c_y))
  .call(g => g.select('.domain').remove())
  .call(g => g.append('text')
    .attr('x', -c_marginLeft)
    .attr('y', 10)
    .attr('fill', 'currentColor')
    .attr('text-anchor', 'start')
    .text('↑ Frecuencia'));
```

Figura 15: Código correspondiente a estructura de gráfico inicial con D3 (5 de 5)

La quinta y última parte se corresponde a la creación de ejes y adición de texto. En la Figura 15, podemos observar la creación primero del eje X y luego del eje Y, facilitada por funciones que proporciona D3 a este efecto.

Al ver este repaso de la creación de un gráfico básico puedo mostrar mejor que esta sintaxis es de un enfoque a bajo nivel, sobre todo comparado con las librerías anteriores de Vega-Lite y Plot. Sin embargo, aunque es más difícil de aprender y acostumbrarse a ella, ofrece un grado de personalización muy superior.

Como se ha visto a lo largo de las Figuras 11-15, puedes cambiar lo que quieras, desde lo más básico, como dimensiones y colores, hasta detalles como el anclaje del texto, se puede cambiar todo al gusto.

Tras realizar estas representaciones gráficas básicas, quise profundizar más y empezar con los gráficos interactivos. Sin embargo, el corpus que estaba usando en el momento no daba mucho juego, así que tuve una reunión con mi tutor para tratar este problema.

4.4 Proyecto agnóstico

Con la librería adecuada elegida y las representaciones tal y como las quería, acabé deseando que el corpus elegido inicialmente proporcionara más datos para crear gráficos más complejos y con mayor interactividad. Tras usar (y modificar) varios corpora sin que ninguno me convenciera, mi tutor me planteó la opción de convertir el proyecto en agnóstico del corpus, es decir, será el usuario el que lo proporcione.

De esta forma, con la posibilidad de usar varios corpora, podía ofrecer todo tipo de representaciones al usuario y que él eligiera la más adecuada para su corpus.

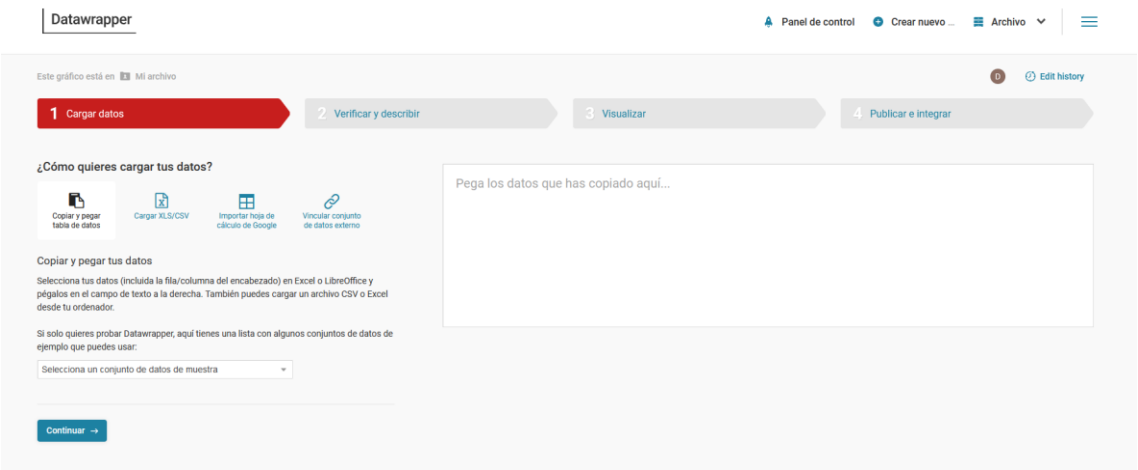

Figura 16: Página de creación de gráficos de Datawrapper

Con este nuevo enfoque y teniendo páginas como Datawrapper [13] de guía (se puede ver la página de creación de gráficos en la Figura 16) surgió la necesidad de dividir el proceso de representación de datos en varios pasos:

- Carga de datos: el usuario puede elegir el archivo almacenado en su propio equipo con el que se va a trabajar (más tarde se implementó la posibilidad de elegir un conjunto de datos de muestra, que son los corpora con los que he trabajado a lo largo del proyecto).

- Verificación: se le presenta una tabla al usuario con la que puede asegurarse de que los datos que ha proporcionado se han procesado correctamente, pudiendo modificarlos de no ser así.

- Visualización: se ofrece un conjunto de gráficos al usuario con el que poder representar visualmente su conjunto de datos, pudiendo elegir las características de los datos a representar, límite de datos y demás, dependiendo del tipo de gráfico.

Con un cambio tan grande en la aplicación, decidí ir abordando los pasos uno por uno, y aunque hubo modificaciones en cada uno de ellos más tarde, voy a explicar el desarrollo de cada uno de ellos por separado en los siguientes apartados.

4.5 Carga de datos

El primer paso para la representación del corpus es la carga de estos, y ahora que el proyecto es agnóstico del corpus, será el usuario el que lo proporcione.

Observable Framework ofrece Inputs muy completos que permiten implementar una subida de archivos de forma sencilla. En mi caso, he decidido que los tipos de archivo disponibles para subir como corpus sean TSV, CSV y JSON ya que la mayoría de los corpora que he encontrado y con los que he trabajado a lo largo del desarrollo de este proyecto presentan estos formatos.

Como ya he mencionado brevemente antes, decidí incorporar la posibilidad de elegir datos de muestra (similar a la de Datawrapper) en la que he incorporado varios corpora con los que he trabajado durante la implementación y que son muy completos para garantizar el buen uso de la herramienta. Todos ellos son muestras que proporciona English-Corpora [14] de forma gratuita del corpus COCA (Corpus of Contemporary American English). Estas muestras son:

- Word Frequency [15]: contiene las 5.000 palabras más usadas del corpus COCA, incluye columnas de *rank* (rango, de 1 a 5000), *lemma* (la palabra), *PoS* (tipo de palabra), *freq* (frecuencia), varias columnas de frecuencia por tipo de texto (blog, hablado, académico…), etc.

- N-grams [16]: contiene los 50.000 n-gramas (conjuntos de n palabras, en este caso 5) más usados cuya primera palabra empieza por ‘m’. Tiene una columna por palabra y otra para la frecuencia del n-grama.

- Collocates [17]: contiene los colocados (palabras que se usan cerca de otra palabra) más frecuentes para una de cada diez palabras, según el ranking de Word Frequency (empezando desde la posición 15, de 10 en 10).

Gracias a la forma en que funciona el Framework, el usuario puede cambiar el corpus que ha subido o elegir uno de los propuestos en cualquier momento y los datos se actualizarán internamente.

````
<!-- Modo de carga -->

```js
// Input archivo local
const archivoLocalInput = Inputs.file({
  label: "Selecciona tu archivo local",
  accept: ".tsv,.csv,.json",
  width: 310,
});
const archivoLocal = Generators.input(archivoLocalInput);

// Selector datos de muestra
const selectorDMInput = Inputs.select(["", "Ngrams (COCA)", "Collocates (COCA)", "Word Frequency (COCA)", "Ngrams (Modificado)"], {label: "Selecciona un conjunto de datos", value: ""});
const selectorDM = Generators.input(selectorDMInput);
```

```js
// Datos de muestra
const coca_ngrams = FileAttachment("./data/coca_ngrams.tsv").tsv();
const collocates = FileAttachment("./data/collocates.tsv").tsv();
const ngrams = FileAttachment("./data/ngrams.json").json();
const wordFrequency = FileAttachment("./data/wordFrequency.tsv").tsv();
```
````

Figura 17: Código correspondiente a la carga de datos

Veremos la pantalla de carga más adelante, aunque sí que quería mostrar lo sencillo que es el código referente a esta parte, mostrado en la Figura 17.

Las dos posibilidades que se ofrecen en la carga de datos tendrán su Input correspondiente, ambos de la librería Observable Inputs de Observable, que nos facilita mucho las cosas, ya que nos permite definir los archivos internamente como constantes globales, accesibles desde cualquier bloque de código de la aplicación.

También se puede ver en la Figura 17 la carga de los archivos que contienen los corpora de muestra, muy sencilla también gracias a la función FileAttachment proporcionada por Markdown. Más tarde veremos la estructura del proyecto y donde se almacenan estos archivos.

Antes de continuar con el apartado de Verificación, quería también mostrar como se trata internamente el parseo de archivos en la Figura 18.

```
```js
// Parseo de archivo
var arch = [{}];
var arch1 = [{}];
var arch2 = [{}];

// Muestra
if(selectorDMInput.value != "") {
  if(selectorDM == "Ngrams (COCA)") {
    arch2 = coca_ngrams;
  } else if(selectorDM == "Collocates (COCA)") {
    arch2 = collocates;
  } else if(selectorDM == "Word Frequency (COCA)") {
    arch2 = wordFrequency;
  } else if(selectorDM == "Ngrams (Modificado)") {
    arch2 = ngrams;
  }
}

// Local
if(archivoLocalInput.value != undefined) {
  if(archivoLocal.name.split('.')[1] == 'tsv') {
    arch1 = archivoLocal.tsv();
  } else if(archivoLocal.name.split('.')[1] == 'csv') {
    arch1 = archivoLocal.csv( {typed: true} );
  } else if(archivoLocal.name.split('.')[1] == 'json') {
    arch1 = archivoLocal.json();
  }
}

if(carga == 1) {
  arch = arch1;
} else if(carga == 2) {
  arch = arch2;
}

const archivo = arch;
```
```

Figura 18: Código correspondiente al parseo de datos.

En la primera parte se identifica el corpus de muestra dependiendo de la opción elegida por el usuario y en la segunda se parsea el archivo subido por el usuario dependiendo del tipo que sea. Cabe destacar que internamente todos los archivos se almacenan en formato JSON.

Luego se elige una u otra opción dependiendo del modo de carga que elija el usuario. He visto necesario hacerlo así ya que, como he mencionado anteriormente, para que una variable sea accesible por otros bloques de código debe de ser una constante. Por tanto,

al asignarle a la constante de archivo el valor de una variable, si el valor de la variable cambia, el valor de la constante también, ejecutando de nuevo todos los bloques de código que la referencien.

4.6 Verificación

Cuando un usuario elige subir su propio corpus, este puede contener errores o puede haber ocurrido un error en el procesamiento de los datos. Por ello, he creído conveniente incluir un paso de verificación, en el que se le muestra una tabla al usuario en la que se le señalarán el tipo de dato y los errores, si hubiera.

La tabla creada es una tabla de HTML y, aunque no voy a mostrar toda su creación, me parece interesante mostrar el modo en que se ha implementado la posibilidad de modificar cada celda si así lo desea lo usuario, de esta forma se pueden corregir errores tanto en el propio corpus como de procesamiento.

```
474     // Celda editable
475     td.onclick = function() {
476       // Checkea si ya está clickada
477       if (this.hasAttribute('data-clicked')) {
478         return;
479       }
480
481       this.setAttribute('data-clicked', 'yes');
482       this.setAttribute('data-text', this.innerText);
483
484       // Input
485       var input = document.createElement('input');
486       input.setAttribute('type', 'text');
487       input.value = this.innerText;
488       input.style.width = (this.offsetWidth - 5) + 'px';
489       input.style.height = (this.offsetHeight - 5) + 'px';
490       input.style.border = '0px';
491       input.style.fontFamily = 'inherit';
492       input.style.fontSize = 'inherit';
493       input.style.textAlign = 'inherit';
494       input.style.backgroundColor = 'LightGoldenRodYellow';
495
```

Figura 19: Código correspondiente a la modificación de celdas (1 de 3)

En la Figura 19 se muestra como una vez creada cada celda se le añade un evento que se activará cuando el usuario haga clic en ella. Lo primero que hará será crear un input para que el usuario introduzca texto, no sin antes comprobar si esta celda ya está siendo editada en ese momento.

```
496         input.onblur = function() {
497           var orig_text = td.getAttribute('data-text');
498           var curr_text = this.value;
499 
500           if (orig_text != curr_text) { // Hay cambios
501             td.removeAttribute('data-clicked');
502             td.removeAttribute('data-text');
503 
504             // Modificar objeto
505             var new_item = structuredClone(item);
506             new_item[keys[i]] = curr_text;
507             archivo[archivo.indexOf(item)] = new_item;
508 
509             // Modificar celda
510             td.innerText = curr_text;
511             item = new_item;
512             td.style.cssText = 'padding: 5px';
513           } else { // No hay cambios
514             td.removeAttribute('data-clicked');
515             td.removeAttribute('data-text');
516 
517             // Modificar celda
518             td.innerText = orig_text;
519             td.style.cssText = 'padding: 5px';
520           }
521 
522           // Color
523           if(isNaN(curr_text) && (keysTypes[i] === 'text')) { // Texto
524             td.style.color = 'seagreen';
525           } else if (!isNaN(curr_text) && (keysTypes[i] === 'number')){ // Número
526             td.style.color = 'royalblue';
527           } else { // Error
528             td.style.color = 'darkred';
529             td.style.backgroundColor = 'lightpink';
530           }
531         }
532 
```

Figura 20: Código correspondiente a la modificación de celdas (2 de 3)

En la Figura 20 se muestra lo que ocurre cuando la celda pierde el foco del usuario, es decir, cuando ha dejado de modificarla. Se comprobará si ha habido un cambio en el contenido de la celda, en cuyo caso se modificará el elemento correspondiente del corpus y la tabla. Tanto si se ha modificado la celda como si no, se volverá a dar color a la celda (no se ha mostrado, pero, a la hora de crear las celdas, se colorean dependiendo del tipo de contenido o si contienen un error).

```
533         // Guardar cambios con tecla Enter
534         input.onkeypress = function(e) {
535           if (e.keyCode == 13) {
536             this.blur();
537           }
538         }
539
540         this.innerText = '';
541         this.style.cssText = 'padding: 0px 0px';
542         this.append(input);
543         this.firstElementChild.select();
544       }
```

Figura 21: Código correspondiente a la modificación de celdas (3 de 3)

Por último, vemos en la Figura 21, la opción añadida de guardar cambios al pulsar la tecla Enter y se deshacen unos cambios de estilo realizados al añadir el input a la celda.

También se le da la posibilidad al usuario de ordenar los datos en la tabla por cada una de las columnas, de forma tanto ascendente como descendente. Esto se ha realizado con Inputs de Observable, similares a los ya mostrados anteriormente, por lo que no veo conveniente repasar esta parte del código.

4.7 Visualización

Último paso, en el que el usuario puede elegir entre hasta 6 tipos de gráficos para representar de forma visual los datos de su corpus.

Mostraré cada uno de ellos en el apartado de Descripción del Producto Final, aunque en este apartado quiero explicar por qué se han elegido estos gráficos, así como algunos fragmentos de código destacables de cada uno de ellos:

- Gráfico de barras: clásico gráfico de barras con identificador en el eje X y valor en el eje Y. Pensado para comparar valores con pocos elementos o ver distintas características para unos mismos elementos (se ha incorporado una barra de desplazamiento por si se elige representar muchos elementos).

El código de este gráfico es bastante similar al mostrado como ejemplo anteriormente, por lo que voy a mostrar algunas consideraciones generales para todos los gráficos.

```
794  // Elige la key
795  const c_key = selectorId;
796
797  // Agrupar por identificador
798  const c_grouped = [];
799  c_data.forEach((item) => {
800    var groupedO = c_grouped.find(o => { return o[c_key] == item[c_key]});
801    if (groupedO != undefined) {
802      keysNumber.forEach((key) => {
803        let num = Number(groupedO[key]) + Number(item[key])
804        groupedO[key] = num.toString();
805      });
806    } else {
807      c_grouped.push(structuredClone(item));
808    }
809  });
810
```

Figura 22: Código correspondiente al gráfico de barras (1 de 2)

En la Figura 22 se muestra la forma en que se ha implementado la agrupación de elementos dependiendo del identificador que elija el usuario, anteriormente se agrupaban siempre por el mismo campo, pero se incorporó la posibilidad de elección por parte del usuario de un identificador por lo que fue necesario un cambio.

```
841  // Borra gráfico anterior (si existe)
842  let barrasAnt = document.getElementById('barras');
843  if (barrasAnt != null) {
844    barrasAnt.remove();
845  }
846
847  // SVG
848  const c_svg = d3.select('#graphBarras').append('svg')
849    .attr('id', 'barras')
850    .attr('width', c_width)
851    .attr('height', c_height)
852    .attr('viewbox', [0, 0, c_width, c_height]);
853
```

Figura 23: Código correspondiente al gráfico de barras (2 de 2)

También quiero mostrar la nueva forma de crear el SVG, ya que esto se aplica a todos los gráficos, en la Figura 23. Antes se podía acceder al SVG desde cualquier bloque de código por lo que se renderizaban directamente en su división de HTML correspondiente. Sin embargo, al añadir varios tipos de gráficos se hizo necesario comprobar si el corpus permitía renderizar cada gráfico, haciendo que el SVG dejara de ser visible en otros bloques de código.

Por ello, ahora al crear el SVG, este se adjunta directamente a su división de HTML correspondiente, comprobando antes si hubiera uno ya creado, en cuyo caso lo elimina.

- Diagrama de sectores: típico gráfico circular con el que ver la proporción del valor de un elemento respecto al valor total. Pensado para utilizarse con un número medio o elevado de elementos.

Es un diagrama básico y, aunque cuenta con un generador de arcos al ser un gráfico circular, la parte de código referente a ello es muy sencilla por lo que no se va a mostrar. Sin embargo, se ha implementado una funcionalidad que hace crecer el sector por el que el usuario pasa el ratón y muestra un "*tooltip*" con los detalles de este, y creo relevante mostrar el código referente a esta parte.

```
978   // Tooltip
979   var tt_hoverDiv = d3.select('#graphSector').append('div')
980     .attr('class', 'tooltip-donut')
981     .style('opacity', 0);
982
```

Figura 24: Código correspondiente al diagrama de sectores (1 de 3)

En la Figura 24 se muestra la creación del "*tooltip*", como una división de HTML adjunta a la división del gráfico, inicialmente con opacidad 0 (oculta).

```
996   .on('mouseover', function (d, i) { // Ratón encima del sector
997     // Resaltar sector
998     d3.select(this)
999       .style('fill-opacity', 1)
1000      .transition().duration(500)
1001      .attr('d', tt_hoverArc);
1002    // Mostrar tooltip
1003    tt_hoverDiv.transition()
1004      .duration(50)
1005      .style('opacity', 1);
1006    // Texto
1007    let label = tt_grouped.find(x => x[selectorVs] === (i.value).toString())[tt_key] + ': ' + i.
        value;
1008    tt_hoverDiv.html(label)
1009      // Coordenadas
1010      .style('left', (d3.pointer(d)[0] + document.getElementById('graphSector').
        getBoundingClientRect().x) + 375 + 'px')
1011      .style('top', (d3.pointer(d)[1] + document.getElementById('graphSector').
        getBoundingClientRect().x) + 325 + 'px');
1012  })
```

Figura 25: Código correspondiente al diagrama de sectores (2 de 3)

Durante la creación de cada sector se añade una función que se activará cuando el usuario pase el cursor por encima de dicho sector, mostrada en la Figura 25. Inicialmente los sectores son coloreados con una opacidad inferior, pero al pasar el cursor por encima se establece en 1 y se aumentará el tamaño de dicho sector. También se mostrará el "*tooltip*" con los datos correspondientes.

```
1013        .on('mouseout', function (d, i) { // Ratón sale del sector
1014          // Volver sector a la normalidad
1015          d3.select(this)
1016            .style('fill-opacity', 0.8)
1017            .transition().duration(500)
1018            .attr('d', tt_arc);
1019          // Ocultar tooltip
1020          tt_hoverDiv.transition()
1021            .duration(50)
1022            .style('opacity', 0);
1023        });
```

Figura 26: Código correspondiente al diagrama de sectores (3 de 3)

Por último, se añade una función, también durante la creación de cada sector que devolverá dicho sector a la normalidad cuando el cursor del ratón ya no se encuentre en este, como se muestra en la Figura 26. También se volverá a ocultar el "*tooltip*".

- Mapa de hexágonos: este gráfico muestra la proporción de elementos para un determinado valor. Puede utilizarse con muchos elementos, aunque se deja de distinguir a qué elemento le corresponde cada proporción. Se recomienda su uso cuando los elementos cuentan con alguna característica identificadora que los pueda dividir en grupos.

Este gráfico es sencillo y con código bastante similar en estructura al de los dos anteriores, por lo que no considero necesario destacar ninguna parte de este.

- Proyección solar: gráfico circular similar al diagrama de sectores que muestra proporciones de elementos anidados. Recomendado para tener una visualización general de elementos con muchos anidados.

La proyección solar, junto con los dos gráficos siguientes, son de tipo jerárquico, requieren por lo tanto generar una disposición de datos interna diferente a la que requieren los gráficos sencillos vistos hasta ahora.

```
1147    // Layout
1148    const ng_hierarchy = d3.hierarchy(sb_data)
1149            .sum(d => d.value)
1150            .sort((a, b) => b.value - a.value);
1151    const ng_root = d3.partition()
1152            .size([2 * Math.PI, ng_hierarchy.height + 1])
1153          (ng_hierarchy);
1154    ng_root.each(d => d.current = d);
1155
```

Figura 27: Código correspondiente a la proyección solar (1 de 3)

En la Figura 27 se puede observar cómo crear una estructura de datos jerárquica para este tipo de gráficos.

Este gráfico es de tipo circular, muy parecido al diagrama de sectores. Sin embargo, los sectores tienen "hijos" en algunos casos, en estos casos debe ser posible hacer clic en el sector para explorarlo en más profundidad, esto se logra con el código visible en la Figura 28.

```
1188    // Clickables si tienen hijos
1189    ng_path.filter(d => d.children)
1190        .style('cursor', 'pointer')
1191        .on('click', clicked);
1192
1193    // Title
1194    const ng_format = d3.format(",d");
1195    ng_path.append('title')
1196        .text(d => `${d.ancestors().map(d => d.data.name).reverse().join("/")}\n${ng_format(d.value)}
            `);
1197
```

Figura 28: Código correspondiente a la proyección solar (2 de 3)

También se incluye en esta figura la creación de un título, que mostrará al usuario la ruta seguida hasta el sector en el que deje el cursor del ratón por unos instantes.

```
1219    // Zoom al clickar
1220    function clicked(event, p) {
1221      ng_parent.datum(p.parent || ng_root);
1222
1223      ng_root.each(d => d.target = {
1224        x0: Math.max(0, Math.min(1, (d.x0 - p.x0) / (p.x1 - p.x0))) * 2 * Math.PI,
1225        x1: Math.max(0, Math.min(1, (d.x1 - p.x0) / (p.x1 - p.x0))) * 2 * Math.PI,
1226        y0: Math.max(0, d.y0 - p.depth),
1227        y1: Math.max(0, d.y1 - p.depth)
1228      });
1229
1230      const t = ng_svg.transition().duration(750);
1231
1232      ng_path.transition(t)
1233          .tween("data", d => {
1234            const i = d3.interpolate(d.current, d.target);
1235            return t => d.current = i(t);
1236          })
1237        .filter(function(d) {
1238          return +this.getAttribute("fill-opacity") || arcVisible(d.target);
1239        })
1240          .attr("fill-opacity", d => arcVisible(d.target) ? (d.children ? 0.6 : 0.4) : 0)
1241          .attr("pointer-events", d => arcVisible(d.target) ? "auto" : "none")
1242
1243          .attrTween("d", d => () => ng_arc(d.current));
1244
1245      ng_label.filter(function(d) {
1246          return +this.getAttribute("fill-opacity") || labelVisible(d.target);
1247        }).transition(t)
1248          .attr("fill-opacity", d => +labelVisible(d.target))
1249          .attrTween("transform", d => () => labelTransform(d.current));
1250    }
```

Figura 29: Código correspondiente a la proyección solar (3 de 3)

Tenemos en la Figura 29 la última porción de código que vamos a ver para este gráfico. Contiene una función de vital importancia, ya que es la que nos permitirá explorar en profundidad un sector que tenga "hijos" y volver al sector "padre" del sector actual.

Esta función elige los sectores que serán visibles a partir del nuevo nodo objetivo. Se apoya en otras funciones auxiliares para determinar si una etiqueta debería mostrarse o no, dependiendo del tamaño del sector y para determinar la propia visibilidad de los sectores. Estas funciones también se usan a la hora de representar el propio gráfico, pero son tan sencillas que no creo conveniente mostrarlas.

- Mapa de árbol: gráfico basado en rectángulos que muestra proporciones de elementos anidados. Alternativa al gráfico de proyección solar, aunque se recomienda su uso con un menor número de elementos anidados.

Este gráfico tiene un funcionamiento similar al anterior y las partes destacables del código también son parecidas por lo que no se incluirá ninguna figura referente a este.

- Árbol de vectores: gráfico que muestra las uniones existentes en elementos anidados. Se recomienda su uso cuando las proporciones no son importantes y se quieren ver los distintos elementos anidados, se puede usar con un gran número de anidados ya que se ha incorporado una barra deslizable, aunque se recomienda no expandir demasiados nodos.

Esta última representación es también similar a las dos anteriores, aunque gran parte de la renderización del gráfico se recoge en su función de actualización (muy extensa, pero, de nuevo, en esencia misma idea que los anteriores). Solo quiero destacar que por lo que acabo de mencionar es necesario especificar una configuración inicial para el gráfico, mostrada en la Figura 30.

```
1553    // Configuración inicial
1554    ct_root.x0 = ct_dy / 2;
1555    ct_root.y0 = 0;
1556    ct_root.descendants().forEach((d, i) => {
1557      d.id = i;
1558      d._children = d.children;
1559      if (d.depth && d.data.name.length !== 4) d.children = null;
1560    });
1561
1562    update(null, ct_root);
1563
```

Figura 30: Código correspondiente al árbol de vectores

Tras indicar la configuración inicial se llama a la función de actualización con la raíz de la estructura jerárquica.

Por último, quiero destacar que estos 3 últimos gráficos necesitan un archivo con una estructura específica para funcionar correctamente, para ello se proporciona como datos de muestra un archivo modificado del N-grams que ofrece English-corpora. En cualquier caso, los botones correspondientes a cada gráfico se activan o desactivan dependiendo del corpus para evitar problemas.

4.8 Despliegue de la herramienta

Mediante el despliegue de la aplicación se puede hacer que ésta sea accesible para todo el mundo. Observable Framework, con la ayuda de NPM permite el despliegue de proyectos para que sean accesibles de forma pública.

Con el plan gratuito que ofrecen, se puede desplegar un proyecto pequeño de forma pública y, aunque es suficiente para este proyecto, para un mayor número de usuarios accediendo y mayores prestaciones, sería necesario acceder a un plan mayor o realizar un despliegue independiente del Framework.

En el Anexo I se explica cómo se podría abordar económicamente esta situación.

4.9 Limitaciones

Como he mencionado en el apartado anterior, el plan gratuito de Observable Framework tiene algunas limitaciones y es probable que con un número elevado de usuarios accediendo a la página se viera reducido su rendimiento.

Además, se permite la subida de archivos de hasta 50 MB, por lo que, si un usuario cuenta con un Corpus de un peso mayor, no podría subirlo (aunque para este proyecto es suficiente).

A mayores, me he dado cuenta de que el rendimiento de la herramienta se ve reducido al representar un número elevado de elementos, sobre todo cuando se aumentan el número de elementos anidados en los gráficos de proyección solar, mapa de árbol y árbol de vectores, ya que el número de elementos a representar aumenta exponencialmente.

4.10 Estrategias para mejorar la mantenibilidad del código

Ya que ha sido necesario mantener todo el código en un mismo fichero md, se han implementado las siguientes estrategias para mejorar la mantenibilidad del código, que se pueden observar en las distintas figuras que contienen código a lo largo de esta memoria:

- División del código mediante comentarios para cada paso de la representación.
- Uso de bloques de Javascript para cada tarea, facilitando la modificación individual de cada una y su distinción.
- Uso de variables globales para almacenar la información necesaria para el correcto funcionamiento de varios bloques o funciones, así como para guardar elementos importantes a los que se accede desde varios puntos.

- Uso de nombres significativos en variables locales y comentarios continuos para aclarar posibles dudas en cuanto a código y poder identificar bloques rápidamente.

4.11 Estructura de ficheros de la aplicación

Se ha seguido la estructura para proyectos recomendada en Observable Framework, dando lugar a la siguiente estructura de ficheros:

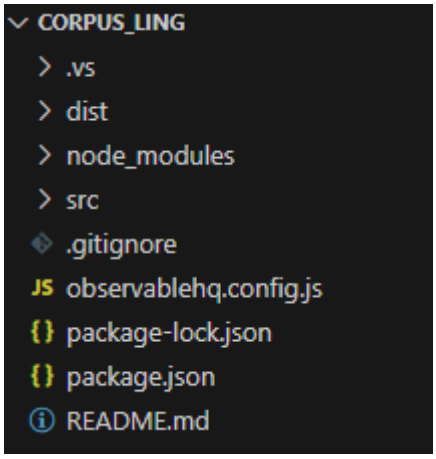

Figura 31: Estructura general de las carpetas del proyecto

En la Figura 31 se ven cuatro carpetas distintas:

- *.vs*: contine datos de Visual Studio Code, generada automáticamente.
- *dist*: carpeta de salida, contiene la página en formato HTML final y algunos activos necesarios para que el sitio funcione, generada automáticamente.
- *node_modules*: contiene las librerías de Javascript del proyecto.
- *src*: carpeta de archivos fuente.

Además de varios archivos de configuración, la carpeta que nos interesa es *src*.

Estructura de src

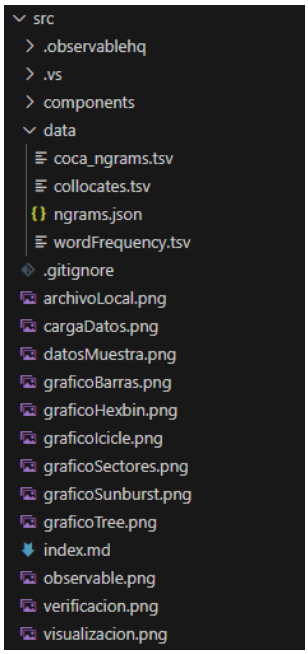

Figura 32: Estructura de archivos de la carpeta src

El directorio *src* tiene, a su vez, varias carpetas:

- *.observablehq*: contine la caché y datos de despliegue, generada automáticamente.
- *.vs*: contine datos de Visual Studio Code, generada automáticamente.
- *components*: contiene módulos compartidos de Javascript, vacía actualmente.
- *data*: contiene cargadores de datos o ficheros estáticos (como se puede ver en la Figura 32, en nuestro caso, los ficheros correspondientes a los datos de muestra).

También contiene index.md, fichero con todo el código de la herramienta y varias imágenes para los distintos botones.

5. Descripción del producto final

En este apartado se va a explicar el funcionamiento de la aplicación. Vamos a ver el resultado final del desarrollo contado a lo largo de esta memoria y las representaciones y el funcionamiento del código descrito hasta ahora.

Primero se accederá a la herramienta a través del siguiente enlace: https://davidprisan.observablehq.cloud/corpusling/.

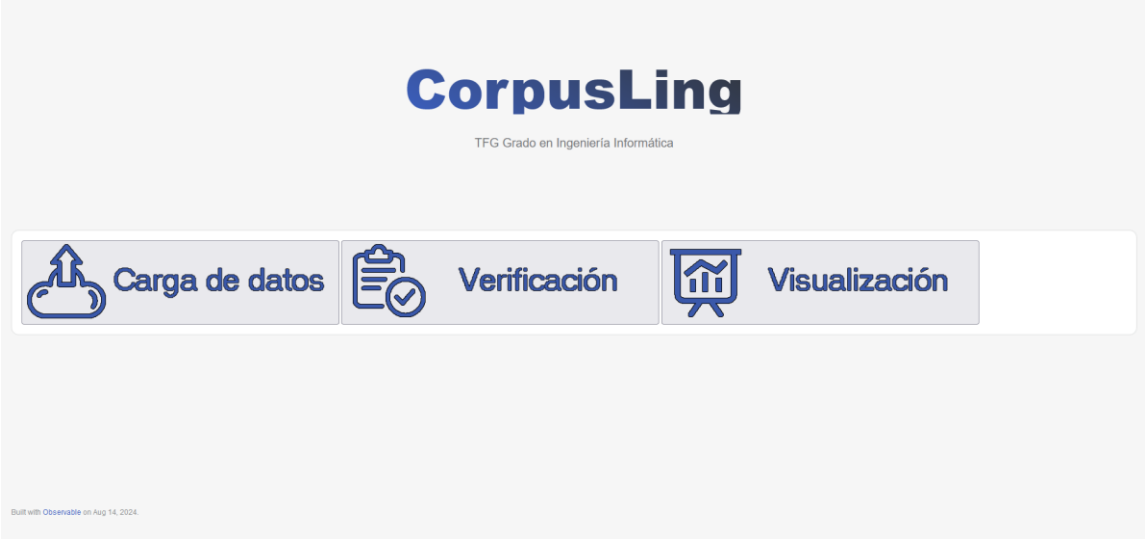

Figura 33: Página principal de la aplicación

Una vez dentro, el usuario verá una página similar a la mostrada en la Figura 33.

Para empezar el proceso de representación de datos, pulsará el botón de Carga de datos, viendo lo mostrado en la Figura 34.

Figura 34: Página de carga de datos

Si el usuario quiere subir un archivo local pulsará el botón de Archivo local y posteriormente el de Examinar archivo, mostrando una ventana similar a la que se muestra en la Figura 35.

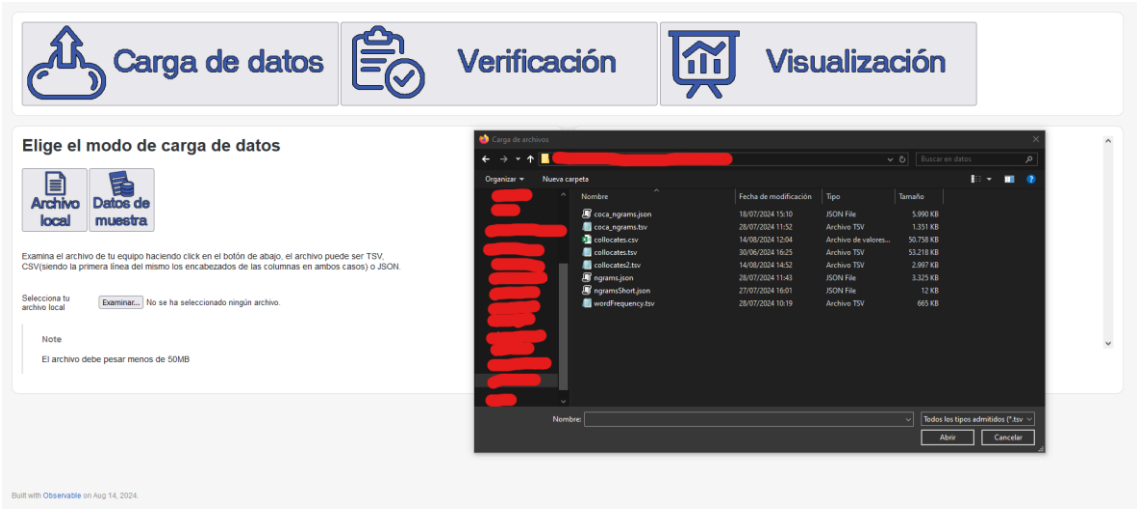

Figura 35: Ventana de examinar archivo

Una vez seleccionado el archivo correspondiente al corpus con el que quiere trabajar verá su archivo procesado en formato JSON de forma similar a lo mostrado en la Figura 36.

Figura 36: Página con archivo examinado

Si, por el contrario, prefiere usar uno de los conjuntos de datos de muestra que proponemos y después de pulsar el botón de Datos de muestra, verá una ventana como la mostrada en la Figura 37.

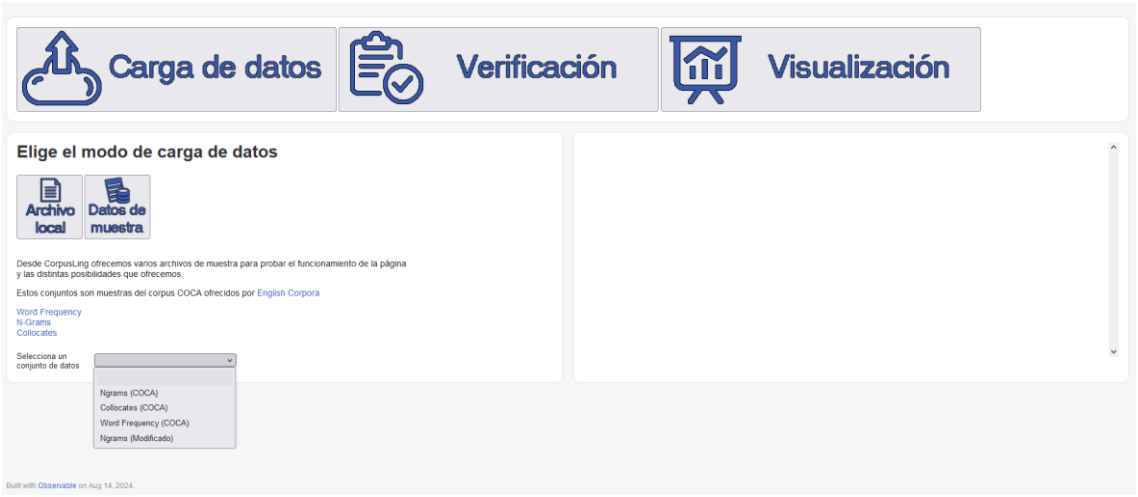

Figura 37: Página de datos de muestra

En esta ventana, el usuario puede elegir el conjunto de datos de muestra que prefiera para trabajar con él, así como acceder a las páginas de English-Corpora de donde proceden estos conjuntos de muestra.

Figura 38: Página con conjunto de datos de muestra elegido

Una vez seleccionado un conjunto de datos de muestra, el usuario puede ver dicho conjunto en formato JSON, de la misma forma que se le muestra un archivo examinado desde su equipo (Figura 38).

Con el conjunto de datos ya elegido, el siguiente paso es la verificación de datos, a la que el usuario pasará una vez pulse en el botón de Verificación, mostrando una página similar a la mostrada en la Figura 39.

Figura 39: Página de verificación

Ahora se le muestra al usuario una tabla con los datos de su corpus, pudiendo ordenar los datos por el valor de cada columna, de forma ascendente o descendente (por defecto: primera columna, ascendente).

Como se le explica al usuario en la página, las columnas se colorearán de verde para datos de tipo texto y de azul para datos de tipo numérico, si hubiera algún error se colorearía la celda correspondiente de rojo, como se puede apreciar en la Figura 40.

Figura 40: Página de verificación con error

Una vez el usuario ha comprobado que sus datos se han procesado correctamente y no hay errores, pasará al apartado de visualización, pulsando el botón correspondiente.

Figura 41: Página de visualización

En una ventana parecida a la de la Figura 41, el usuario podrá elegir el tipo de gráfico con el que quiere representar su corpus pulsando el botón para cada uno de ellos en la botonera.

Vamos a ver ahora como se representa cada uno de los gráficos usando el conjunto de datos de muestra de Word Frequency:

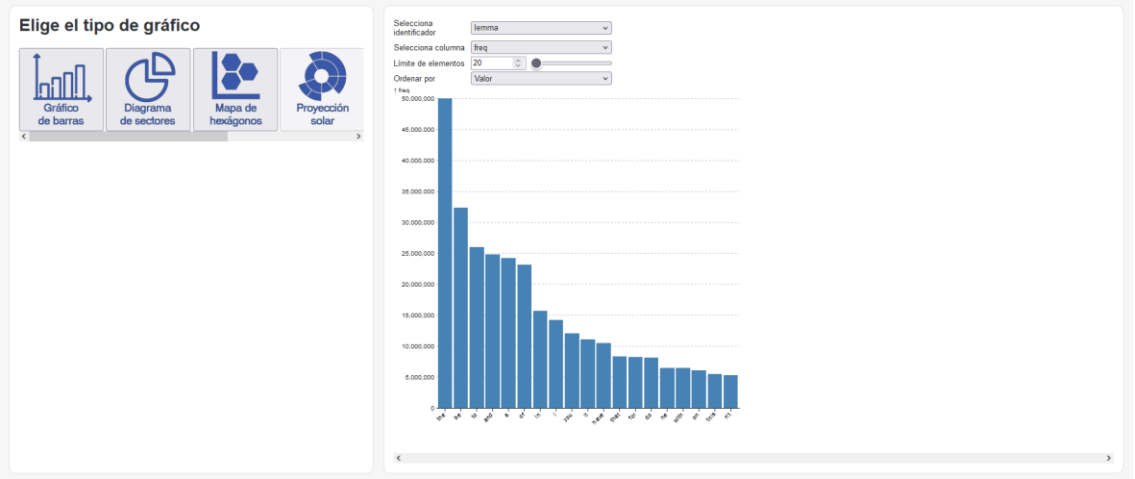

Figura 42: Gráfico de barras

El primero es el clásico gráfico de barras, simple pero eficaz (Figura 42). Se le permite al usuario elegir el identificador de entre las columnas de tipo texto para el eje X y el valor de entre las columnas de tipo numérico para el eje Y.

También se ofrecen las posibilidades de limitar el número de elementos a mostrar y la forma en que se ordenan dichos elementos: por valor o por identificador.

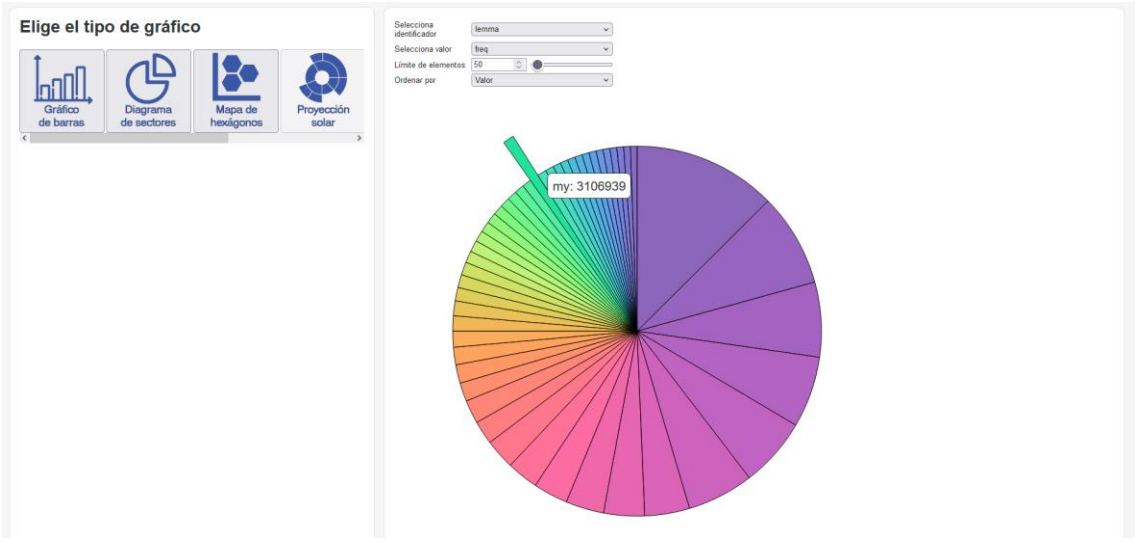

Figura 43: Diagrama de sectores

El siguiente es el gráfico de sectores, ofrece los mismos filtros que el gráfico de barras. Cuando el ratón pasa por encima de un sector, dicho sector se resalta y se le muestra al usuario el identificador y valor del sector, para facilitar la legibilidad. Se puede ver un ejemplo en la Figura 43.

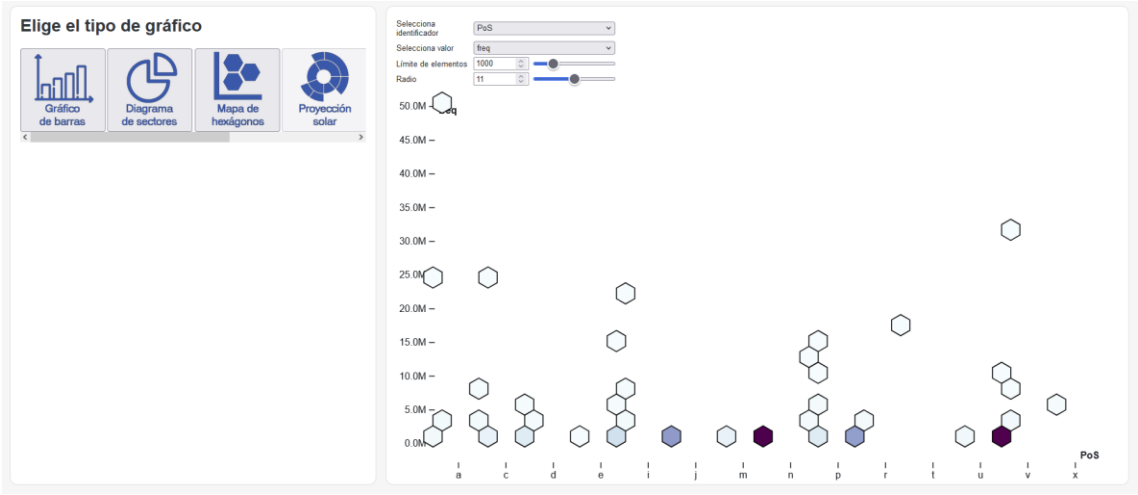

Figura 44: Mapa de hexágonos

El gráfico mostrado en la Figura 44 es un mapa de hexágonos, cuenta con los mismos filtros que los anteriores, a excepción de la opción de ordenar, sustituida por un selector de radio para aumentar o disminuir el tamaño de los hexágonos.

Como se ha mencionado anteriormente, se recomienda el uso de este gráfico si el corpus cuenta con alguna característica que permita agrupar elementos. El corpus de Word Frequency cuenta con la columna de *PoS* (tipo de palabra), por lo que, aunque seleccionemos 1.000 elementos, el gráfico sigue siendo legible.

Cabe destacar que el gráfico se representará aunque la columna seleccionada no agrupe los elementos, pero el eje X no será legible con muchos elementos.

Como se puede apreciar en las Figuras 42, 43 y 44, el resto de los botones están desactivados dado que el corpus no es adecuado para utilizarlos. Para ver los tres gráficos restantes utilizaremos el corpus N-grams (modificado).

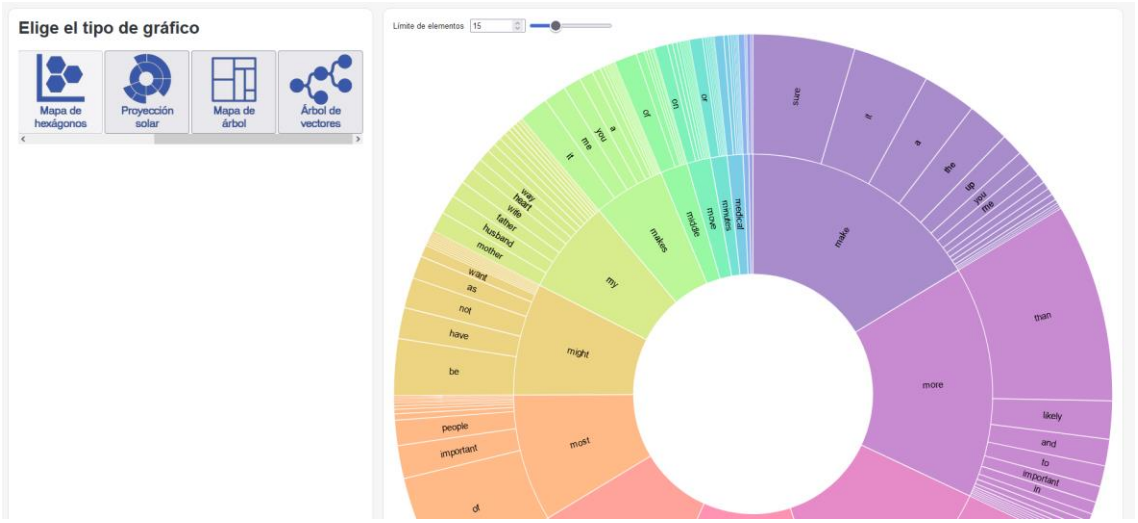

Figura 45: Proyección solar (raíz)

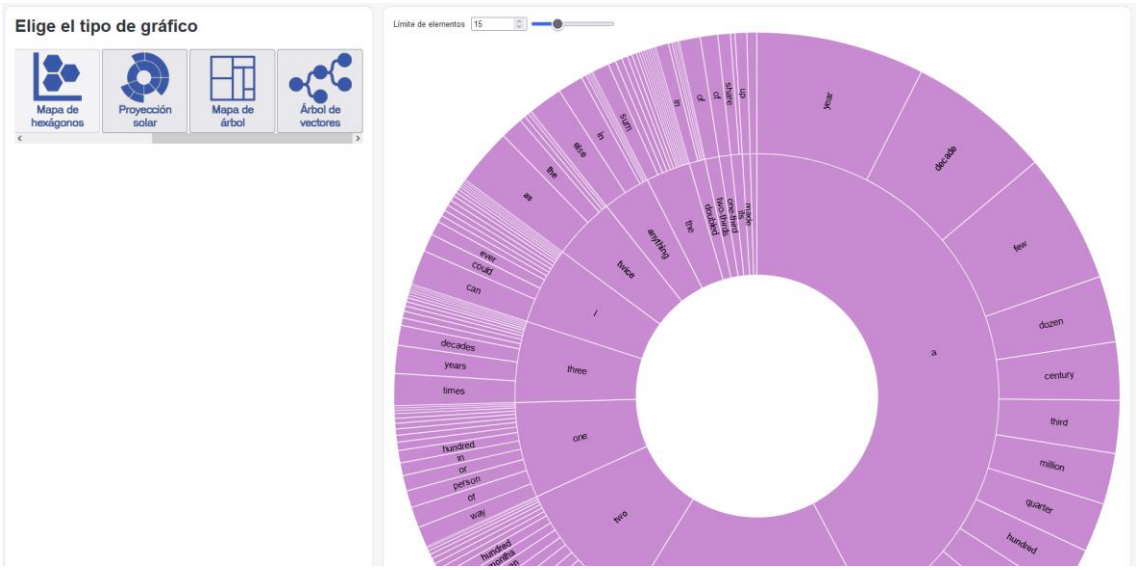

Figura 46: Proyección solar (nodo expandido)

El primer gráfico para elementos anidados es el de proyección solar, en este gráfico se puede ver las distintas proporciones para cada elemento anidado y navegar por la estructura, de forma que al pulsar en un sector nos muestra sus elementos anidados y podemos volver un al nodo padre del elemento actual pulsando el “agujero” en el interior del gráfico.

En las Figuras 45 y 46 se ve primero el gráfico desde la raíz y después los elementos anidados en *more than*. Si el sector es muy pequeño no se muestra el texto, pero se puede saber a qué elemento pertenece dejando el ratón encima.

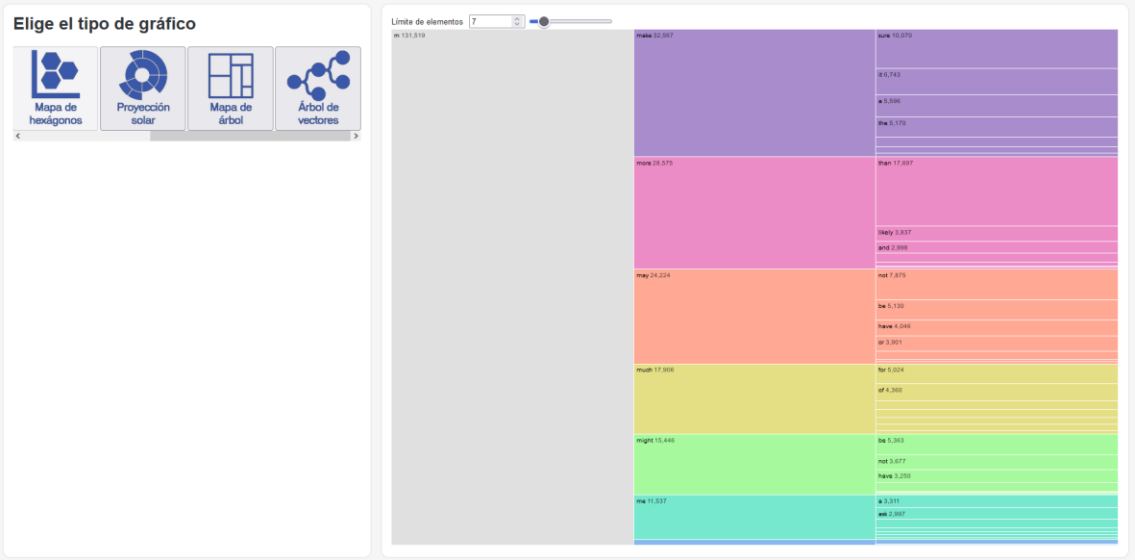

Figura 47: Mapa de árbol (raíz)

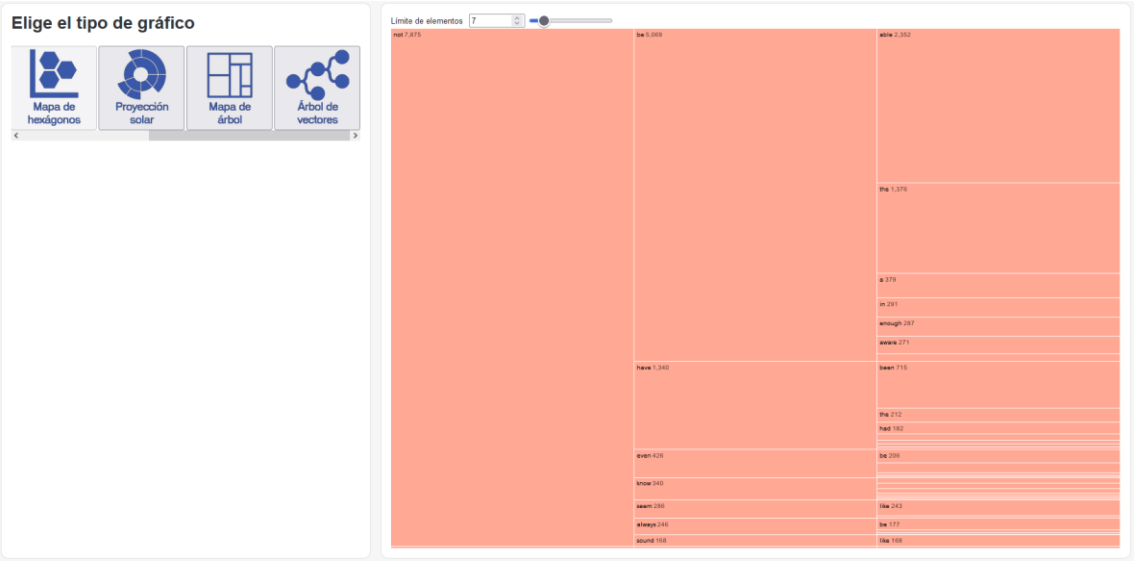

Figura 48: Mapa de árbol (nodo expandido)

El siguiente gráfico es el de mapa de árbol, en este gráfico se puede ver las distintas proporciones para cada elemento anidado y navegar por la estructura, de forma similar que en el gráfico anterior. En este caso, para volver al nodo padre, basta con pulsar en el nodo que nos encontramos ahora (el que está más a la izquierda).

En las dos Figuras 47 y 48 se ve primero el gráfico desde la raíz y después los elementos anidados en *may not*. Si el rectángulo es muy pequeño no se muestra el texto, pero se puede saber a qué elemento pertenece dejando el ratón encima.

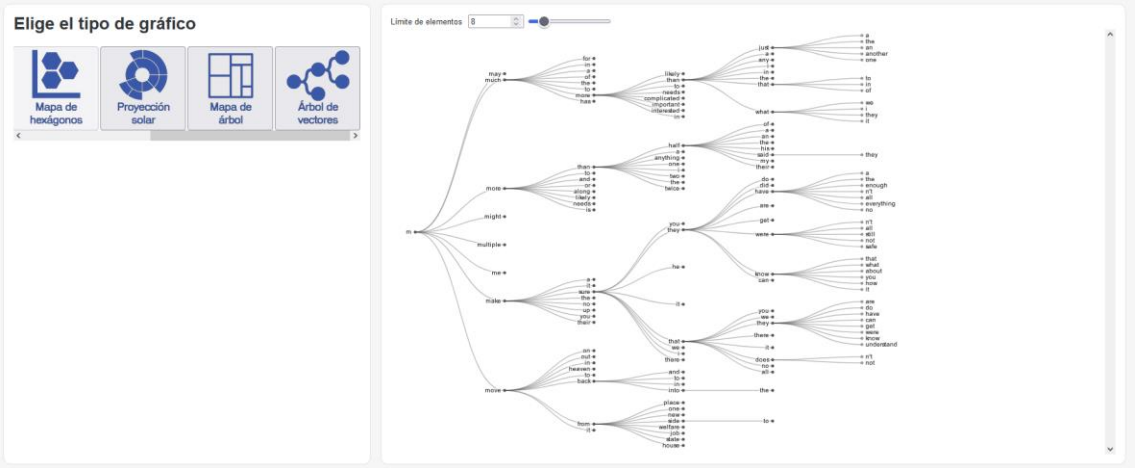

Figura 49: Árbol de vectores

Por último, tenemos el gráfico de árbol de vectores, representado en la Figura 49. En este gráfico no se reflejan las proporciones de los elementos anidados, solo el identificador de cada nodo. Por defecto se expanden aquellos nodos cuyo identificador es de cuatro letras, aunque el usuario puede expandir y colapsar los nodos que desee pulsando en ellos.

6. Conclusiones y líneas de trabajo futuras

Se ha desarrollado una aplicación web que permite representar corpora visualmente de manera remota, accesible desde cualquier equipo.

Desde el punto de vista del desarrollo y gracias a Observable Framework, la aplicación es fácil de mantener y de realizar nuevos incrementos que introduzcan nuevas funcionalidades o aporten un valor añadido al proyecto. Adicionalmente, la aplicación presenta un alto grado de escalabilidad, en caso de que hubiera aumento considerable en el número de usuarios que utiliza la aplicación.

Para desarrollar la aplicación se ha utilizado la metodología ágil SCRUM, que ha permitido desarrollar la aplicación de manera iterativa e incremental.

Realizar este proyecto ha supuesto todo un reto, ya que, aunque estaba familiarizado con Java, he investigado a fondo varias librerías de Javascript hasta encontrar la adecuada. Además, no había desarrollado mucho en web hasta ahora.

Gracias a esto, he podido aprender muchísimo sobre nuevas tecnologías y herramientas que conocía, por lo que creo que en general ha sido una experiencia muy positiva.

En general, creo que he aprovechado al máximo la oportunidad que he tenido para aprender y siempre he estado abierto a investigar nuevas herramientas, probar diferentes alternativas y hacer cambios que supongan una mejora en la aplicación.

En cuanto a las ampliaciones futuras, se podrían introducir muchos otros gráficos, aunque creo que los que mejor pueden representar un corpus son los que he incluido, pero cuanta más variedad, mejor.

También se podrían añadir otras funcionalidades, como la posibilidad de, al crear una representación, poder publicarla e incorporarla en otras páginas, o guardar tus representaciones implementando gestión de usuarios, pero se ha decidido desarrollar una página simple, accesible a todo el mundo y que no requiera de registro ni de inicio de sesión para su uso. Así mismo, destacar que, como se le indica al usuario, si sube un corpus para trabajar con él, en cuanto cierre la página se eliminará de la aplicación y no se conservará ningún dato.

Otra característica que estuvo en el tintero y quería implementar es NLP (Procesamiento de Lenguaje Natural), aunque por falta de tiempo no he podido añadir. Con esta funcionalidad se conseguiría ampliar aún más el horizonte de la aplicación, ya que permitiría tanto la subida de textos escritos sin formatear como la "creación" de nuevos corpora a partir de otros ya existentes.

7. Bibliografía

[1] GitHub: https://github.com/

[2] Visual Studio Code: https://code.visualstudio.com/

[3] Observable Framework: https://observablehq.com/framework/

[4] Markdown: https://markdown.es/

[5] HTML: https://html.spec.whatwg.org/multipage/

[6] Javascript: https://www.java.com

[7] Node.js: https://nodejs.org/en

[8] NPM: https://www.npmjs.com/

[9] D3.js: https://d3js.org/

[10] CORPES: https://apps2.rae.es/corpes/

[11] Vega-Lite: https://vega.github.io/vega-lite/

[12] Observable Plot: https://observablehq.com/plot/

[13] Datawrapper: https://www.datawrapper.de/charts

[14] English-Corpora: https://www.english-corpora.org/

[15] Word Frequency: https://www.wordfrequency.info/

[16] N-grams: https://www.ngrams.info/

[17] Collocates: https://www.collocates.info/

[18] StackOverflow: https://stackoverflow.com/

[19] Galería D3: https://observablehq.com/@d3/gallery

[20] Galería Plot: https://observablehq.com/@observablehq/plot-gallery

[21] Sönning, Lukas & Ole Schützler. 2023. "Data visualization in corpus linguistics: Critical reflections and future directions". *Data visualization in corpus linguistics: Critical reflections and future directions* (Studies in Variation, Contacts and Change in English 22), ed. by Ole Schützler and Lukas Sönning. Helsinki: VARIENG: https://urn.fi/URN:NBN:fi:varieng:series-22-0

[22] Kaggle: https://www.kaggle.com