SimMapper
A Tool for Creating Similarity Maps from Multidimensional Datasets

Group 3

Jihad Itani, Piotr Kupiec, Emanuel Moser and Martin Sackl

706.057 Information Visualisation SS 2021
Graz University of Technology

01 July 2021

Abstract

Too many input dimensions of a given dataset can make analysis and visualization challenging. Dimensionality reduction refers to a set of widely used techniques to reduce the number of input dimensions in a dataset. SimMapper is a tool using DruidJS, which is a rather new JavaScript library for dimensionality reduction. With this application, multiple dimensionality reduction methods can be applied in order to project high-dimensional data to a lower dimensionality while keeping method-specific properties of the data. Users are able to customize visualizations, select dimensions, and export plots.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

For this project, we were tasked to build a standalone application, which provides a graphical user interface around the dimensionality reduction library DruidJS [Cutura et al. 2020; Cutura and Rivière 2021]. The idea was that the user is able to select an arbitrary dataset in CSV format, select which dimensions should be included, and create a plot with a chosen dimensionality reduction method. In addition, it should be possible to customize the style of the visualization with glyphs of different color, shape, and size. Finally, the user should be able to export the visualization in SVG format, preferably responsive and human-readable. In the following sections, we will briefly describe what dimensionality reduction is and which technologies were used in the creation of SimMapper.

1.1 Dimensionality Reduction

Dimensionality reduction refers to widely used set of techniques which reduce the number of dimensions in a dataset. In machine learning and data science, dimensionality reduction significantly lowers the number of features, while preserving the relevant information in the data as accurately as possible in order to still be able to learn from it and make predictions. If number of dimensions is high, it is hard to visualize a dataset and work on it.

The most basic algorithm for dimensionality reduction is called Principal Component Analysis (PCA) [Pramoditha 2021]. It is a linear dimensionality reduction algorithm, which transforms a set of variables (dimensions) into a smaller number of unit vectors called principal components. In the process the algorithm tries to retain as much of the relevant information found in the original dataset as possible.

1.2 Project Setup

In the following section, we will briefly introduce the technologies and frameworks that were used to create SimMapper. Some frameworks (namely DruidJS and TypeScript) have been given as requirements. Additionally, it was a requirement that we will build a web app rather than a website, which can also run as a standalone client and be used completely offline.

Furthermore, since we are hosting our code currently on GitLab [Itani et al. 2021b], we configured a pipeline that will generate all necessary files and push them to our GitLab page [Itani et al. 2021a], to produce a hosted web version of the application.

1.2.1 ElectronJS

Electron or ElectronJS is a framework to create a desktop application based on web technologies, mainly JavaScript, HTML and CSS [OpenJS 2017]. We chose it because we were all fairly proficient with the aforementioned technologies. In addition, Electron offers cross-platform capabilities, which mean that this app could be built and run on Windows, Linux, and MacOS. Finally, it also supports the use of TypeScript as of June 2017 [Sikelianos 2017], which was a requirement to be used.

1.2.2 TypeScript

“TypeScript is an open-source programming language created by Microsoft. It’s a superset of JavaScript that extends the language by adding support for static types.” [Microsoft 2017]. It was a requirement to use TypeScript, because it introduces static types to JavaScript. In addition, it gave us the vast number of JavaScript packages at our disposal with the benefits TypeScript introduced.

1.2.3 Gulp

Gulp [Gulp 2021] was used as a task runner, to efficiently import all of our JavaScript libraries and compile them, clean up of our project files, and bundle everything together into an executable app.

1.2.4 Bulma

Bulma is an open source CSS framework that provides ready made components that can be easily used to create responsive web pages [Thomas 2021]. For us, it was easy to use and also provided the option that we could overwrite the provided SCSS variables of components, but we decided importing the minified CSS file would be sufficient.

1.2.5 DruidJS

DruidJS [Cutura and Rivière 2021] is a fairly new (published 2020) JavaScript library for dimensionality reduction which offers a wide array of different dimensionality reduction methods and is compatible with the data structure of D3 [Bostock 2021].

1.2.6 D3

D3 [Bostock 2021] is probably the go to library when producing interactive data-driven visualizations. We are using just one of its many modules to work with CSV files. We originally planned to use it to generate the visualizations based on the data DruidJS gives us, but switched later to PlotlyJS.

1.2.7 PlotlyJS

Plotly [Plotly 2021] is a high-level charting library built on top of D3 [Bostock 2021] and stack.gl [Stackgl 2020]. We originally intended to use D3, but creating a plot proved to be quite tedious, so we switched to PlotlyJS. PlotlyJS proved is quite powerful and also includes ready-to-use SVG export functionality.

Chapter 2

Implementation

This chapter describes the three most important aspects of the implementation of SimMapper: data preprocessing, dimensionality reduction, and visualization. The screenshots refered to here are included as part of Chapter 3.

2.1 Data Preprocessing

The user prepares a dataset as a CSV file on his or her device. To open a file, the main screen has a menu item Open Dataset (see Figure 3.1) as well as a centered file input element (if no dataset has been selected so far). After selecting a dataset, a popup (see Figure 3.2) will be shown to the user displaying a preview of the dataset (the first five rows) and two dropdowns, with which the user can set the character encoding (UTF-8, UTF-16, etc.) and the CSV separator (comma, semicolon, tab, etc.). The checkbox above each column allows the user to select which of the dimensions should be included (this can be done afterwards at anytime from the main navigation bar).

The parsing of the CSV dataset is done by the D3 library. A filereader first reads the file's content and then uses the D3 parsing functionality, which depends on the selected CSV separator. Listing 2.1 shows the three options of parsing data. After parsing the data, only the selected dimensions are added to an array, which is then displayed within the above mentioned popup in form of a table. Creating the table is again done by D3 and the popup is created with SweetAlert2 by calling a `Swal.fire()` function. Whenever the user presses the Select dataset button, the dimensionality reduction algorithm is started.

2.2 Dimensionality Reduction

DruidJS comes with great implementations of dimensionality reduction techniques. SimMapper first filters all dimensions to ensure they contain only numerical data. Afterwards a *Druid Matrix* is created with the filtered data. All data values are normalized to the range from 0.0 to 1.0 in advance.

Since the user is able to select from multiple reduction methods, each having different required or optional parameters, the tool displays these parameters to the user whenever he or she changes the reduction method. The selected method and parameters are then used to reduce the dimensionality of the dataset within a *switch case* using Druid's reduction functionality. The case for *TSNE* is shown in Listing 2.2.

2.3 Visualization

After the application has processed and reduced the data, it is ready to visualize it with help of the Plotly library. The area holding the visualization is separated in two halves: the actual plot visualization and a container showing the selected parameters and the reduction method.

```
switch (separator) {
  case ',':
    dat = d3.csvParse(item);
    break;
  case 'Tabulator':
    dat = d3.tsvParse(item);
    break;
  default:
    let psv = d3.dsvFormat(separator);
    dat = psv.parse(item);
}
```

Listing 2.1: Seperator differentiation.

```
case 'TSNE':
    let dim_perplexity: any =
        <HTMLInputElement>document.getElementById('dim_opt_perplexity');
    let dim_perplexity_value: any = dim_perplexity.value;

    let dim_epsilon: any =
        <HTMLInputElement>document.getElementById('dim_opt_epsilon');
    let dim_epsilon_value: any = dim_epsilon.value;

    dr = new druid.TSNE(X,
        dim_perplexity_value,
        dim_epsilon_value,
        2, // dimensions reduced to
        dim_metric,
        dim_opt_seed_value)
    dr = dr.transform().to2dArray
    break;
```

Listing 2.2: TSNE case.

Plotly comes with a variety of functionality and several options for the user to visualize the given data. Since there is no option to plot in a 1:1 ratio, the tool calculates the given `offsetWidth` of the container and sets this as width and height of the plot. It then creates a `layout` object, which defines title, height, yaxis and xaxis. The two axes define the format of the ticks, which is set to `.1f`.

Next, the plot is created by calling the `newPlot` function from Plotly and defining several attributes, shown in Listing 2.3. This function takes the container, in which the plot should be displayed (`plot_col`), the actual data (`data`), the before mentioned layout object (`layout`) and an options object, which is defined directly in the function call. Within this object, the plot is set to responsive, several unnecessary buttons are disabled, and one new button is added. This button makes it possible to export a SVG from the plot. Plotly already comes with such functionality, but since the exported SVG file is not freely scalable by default, SimMapper has its own export function. This function simply reads all the SVG components of the plot and removes the troublesome `width` and `height` attributes. This functionality could be improved in future work, but delivers already a freely scalable SVG.

Plotly also allows users to define several customizable visualization attributes like the type of the plot, mode, and style of the markers. Therefore, SimMapper creates the `data` object shown in Listing 2.4 by setting the `x` and `y` values, choosing the default plot type `scatter` and mode `markers`. It then sets the marker's color, size and symbol. These attributes can be changed by the user by clicking on the Glyph Styling menu item, which displays a pop-up holding corresponding dropdowns and input fields. Whenever

```
Plotly.newPlot(plot_col, data, layout, {
  responsive: false,
  toImageButtonOptions: {
    filename: 'plot_export',
    format: 'svg'
  },
  modeBarButtonsToAdd: [
    {
      name: 'Save as SVG',
      title: 'Save as SVG',
      icon: icon1,
      click: () => exportSVG(true)
    },
  ],
  modeBarButtonsToRemove: [
    'toImage', 'hoverClosestGl2d',
    'hoverClosestGeo', 'hoverClosestCartesian',
    'sendDataToCloud', 'lasso2d', 'zoomIn2d',
    'zoomOut2d', 'resetScale2d', 'toggleSpikelines',
    'hoverCompareCartesian'],
    displaylogo: false,
});
```

Listing 2.3: Plotly visualization.

```
let data: Plotly.Data[] = [{
  x: x,
  y: y,
  type: 'scatter',
  mode: 'markers',
  marker: {
    color: marker_color,
    size: marker_size,
    symbol: marker_shape
  },
}];
```

Listing 2.4: Customization of glyphs in Plotly.

these values are changed and a new visualization is created, they will be applied to the plot.

Chapter 3

Walkthrough

This chapter provides a step-by-step walkthrough of the SimMapper web application from start to finish, with all the details in between that might be of help for future users. Additionally, we also created a short video [Sackl 2021] to showcase SimMapper and its functionality.

3.1 Opening a Dataset

Having opened the application, the user can click on either the button Open Dataset at the top left to select a file, or click in the middle of the application where it also says Open Dataset, as shown in Figure 3.1. Once the user has selected their dataset, they will see a pop-up window displaying the first 5 rows and an indication of how many rows were read in total, as shown in Figure 3.2. Two drop-down menus indicate which character encoding and column separator to use, although for the current build, we only allow for comma-separated files. The check boxes above each column allow the user to choose exactly which dimensions to include.

3.2 Dimensionality Reduction

Once the dataset is selected, and the user is satisfied with their selection of what they want to include for their visualization, we move onto the next step of the application, and that is where we use dimensionality reduction methods on the selected dataset dimensions by the user. There are currently five methods available in SimMapper: PCA, TSNE, ISOMAP, FASTMAP, and MDS. Some of the method take additional parameters, which are shown to the right. Once everything is filled in and the user wants to see how it looks like, they can go ahead and click on the green button Visualize. The reduced dataset is plotted in a graph, with the method and any applied parameters shown on the right.

3.2.1 PCA

The first reduction method, PCA, does not take any additional parameters. It is shown in Figure 3.3.

3.2.2 TSNE

Next, the TSNE method takes four parameters which are the seed, distance function, perplexity, and epsilon, as shown in Figure 3.4. The distance functions available for TSNE are euclidian, chebychev, cosine, euclidian_squared, manhattan, and canberra, as shown in Figure 3.5.

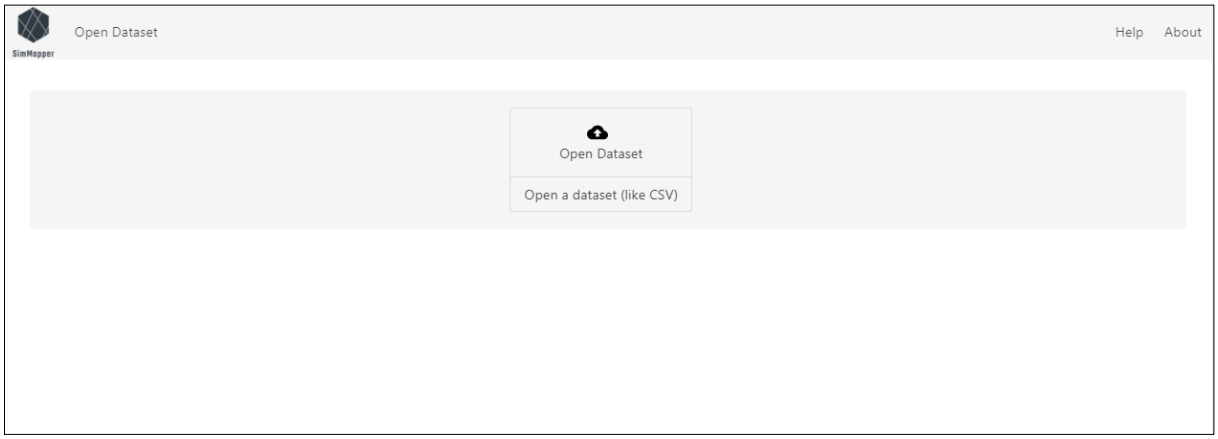

Figure 3.1: Main screen of SimMapper.

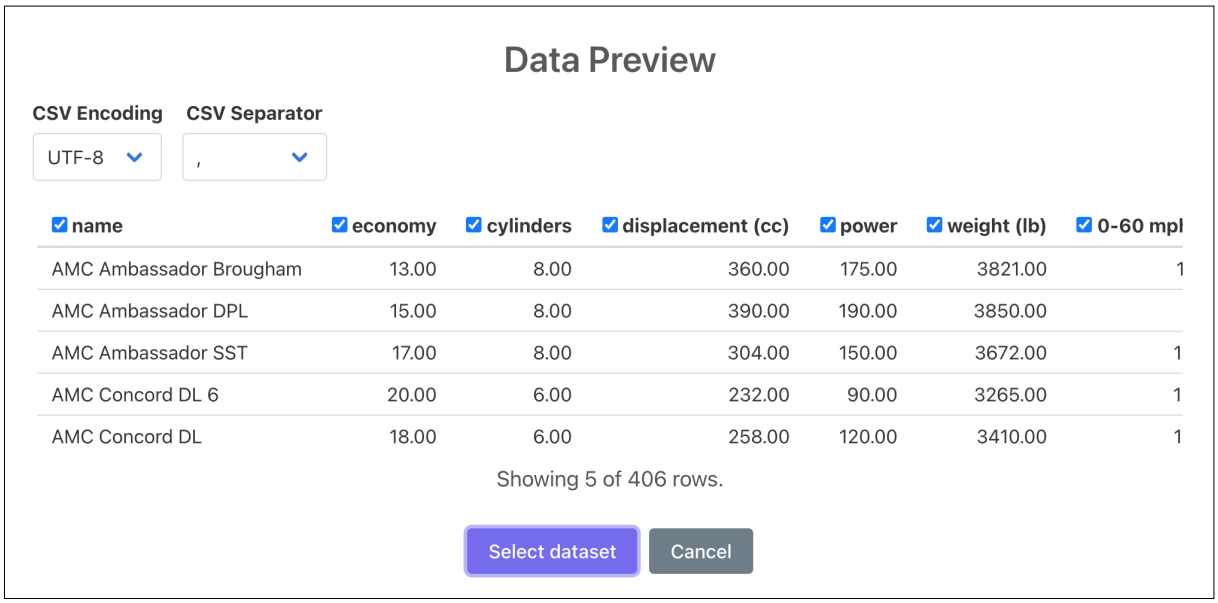

Figure 3.2: Preview of the dataset.

3.2.3 ISOMAP

Thirdly, the ISOMAP method takes three parameters, which are the neighbors (which is required), seed, and a distance function (the same distance functions as for TSNE). You can see an ISOMAP graph in Figure 3.6.

3.2.4 FASTMAP

Fourthly, the FASTMAP method takes only two parameters, which are the seed and the distance function (the same distance functions as for TSNE). You can see an FASTMAP graph in Figure 3.7.

3.2.5 MDS

Finally, the MDS method takes two parameters, which are the seed and the distance function (the same distance functions as for TSNE). You can see an MDS graph in Figure 3.8.

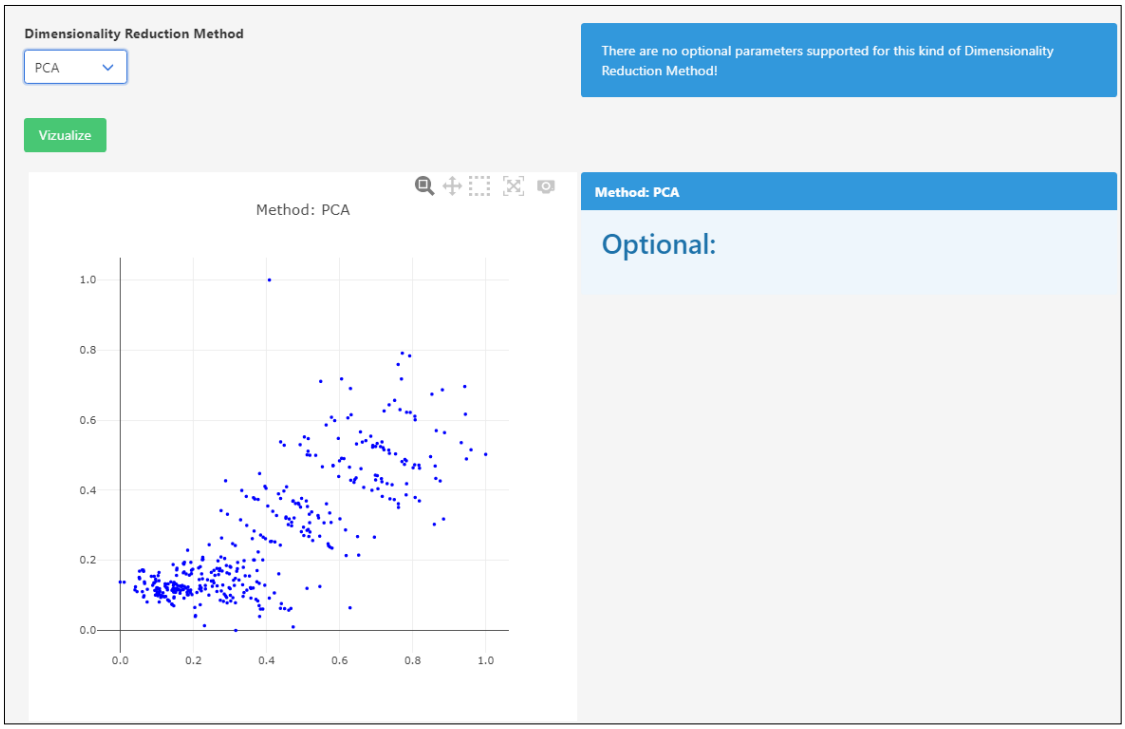

Figure 3.3: Graph produced with PCA.

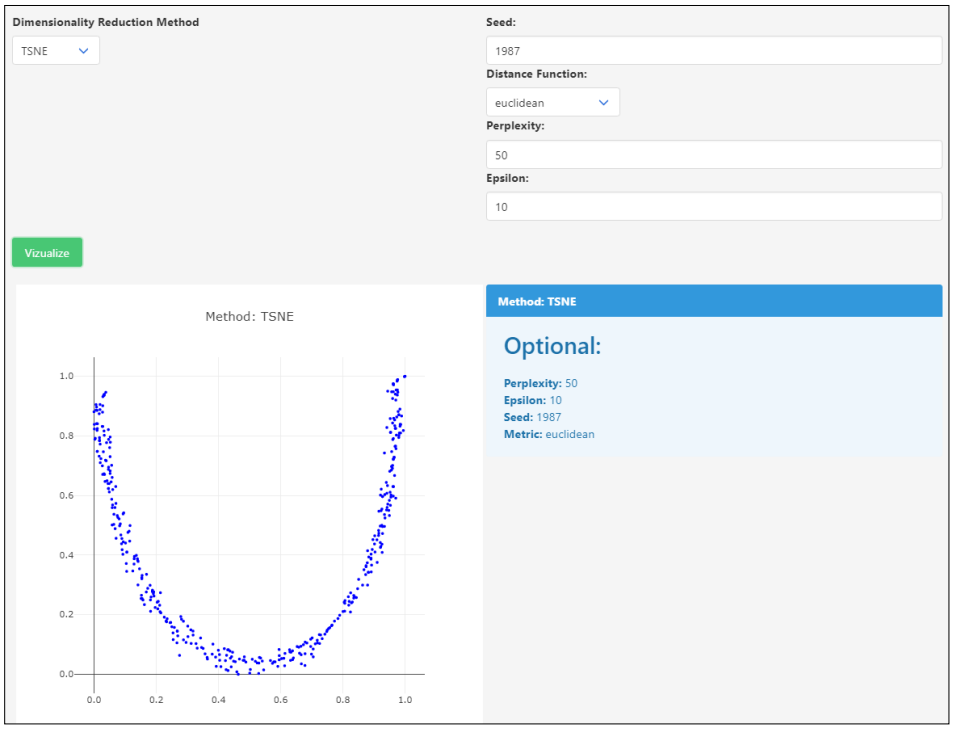

Figure 3.4: Graph produced with TSNE.

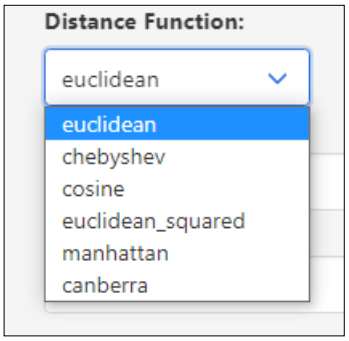

Figure 3.5: Distance functions available.

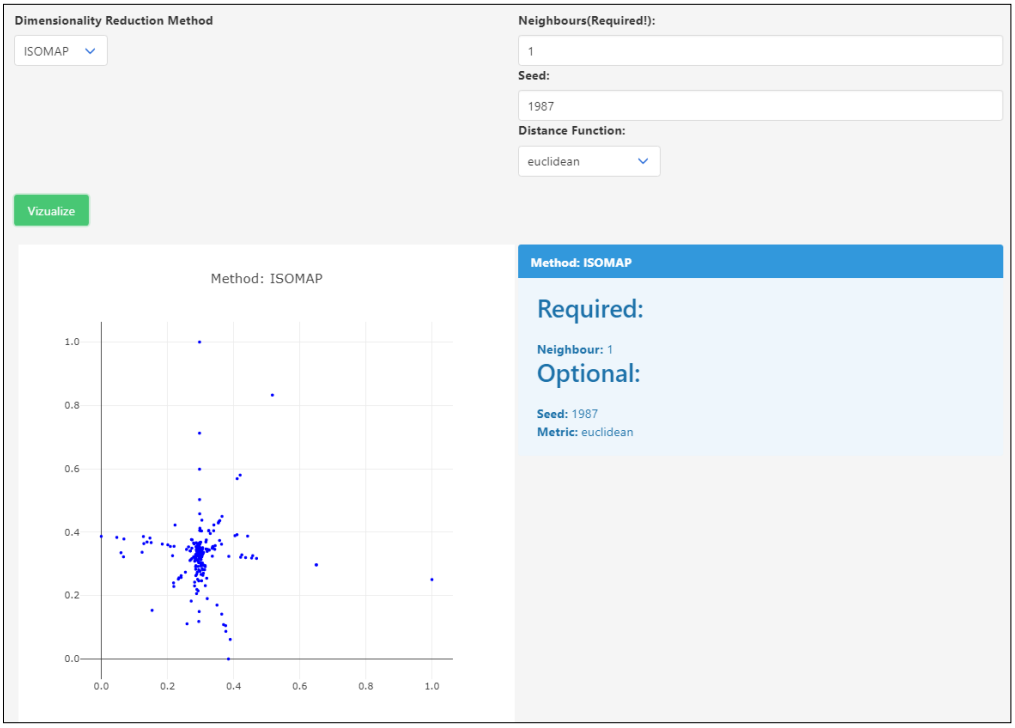

Figure 3.6: Graph produced with ISOMAP.

3.3 Graph Options

At the top right corner of each graph, five iconic buttons are available for further interaction: Zoom In, Pan, Box Select, Autoscale, and finally Save as SVG. They can be seen in Figure 3.9. The Zoom In button allows the user to zoom into the graph. The Pan button allows the user to move around the graph. It is only really useful after the user has zoomed in. Box Select highlights a certain area of the graph for the user. Autoscale resets the previous options and puts the graph back in its original state. Save as SVG lets the user save the graph as an SVG file on their local drive, which is also the same as the Export SVG button shown in Figure 3.10.

3.4 Select Dimensions

At the top of the page, after selecting the dataset and having the graph appear, the user still has the option to click on Select Dimensions, shown at the top of Figure 3.10. The Data Preview window appears and dimensions can be ticked or unticked to add or remove them from consideration, as shown previously in

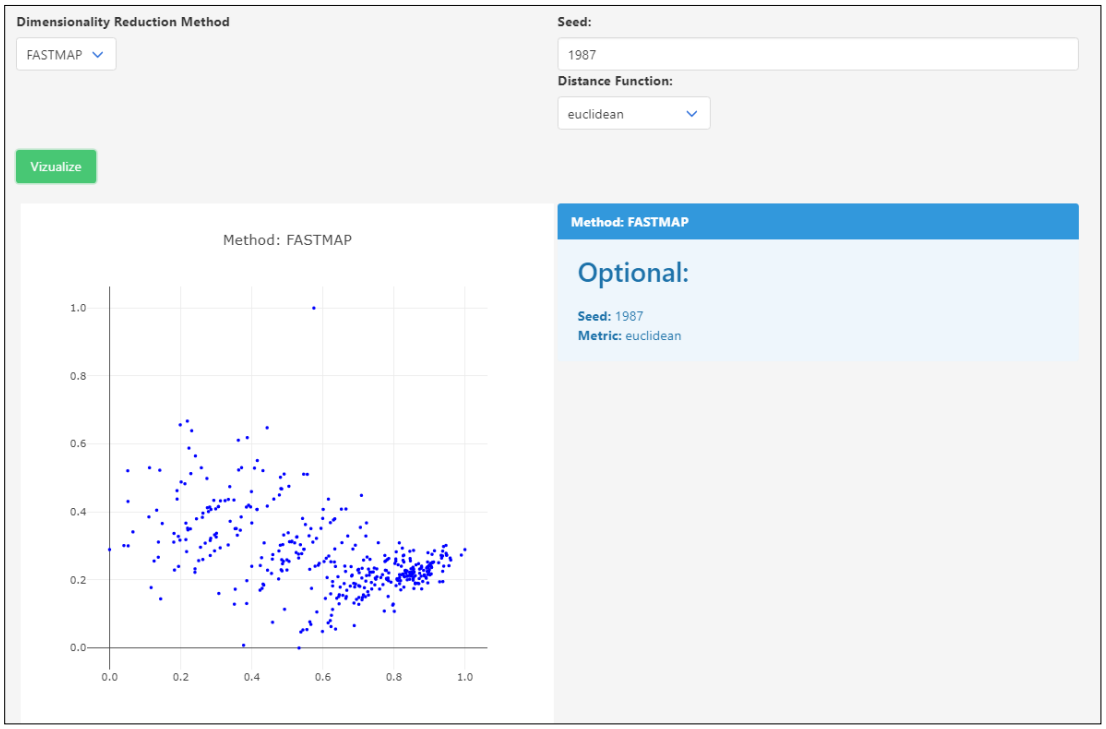

Figure 3.7: Graph produced with FASTMAP.

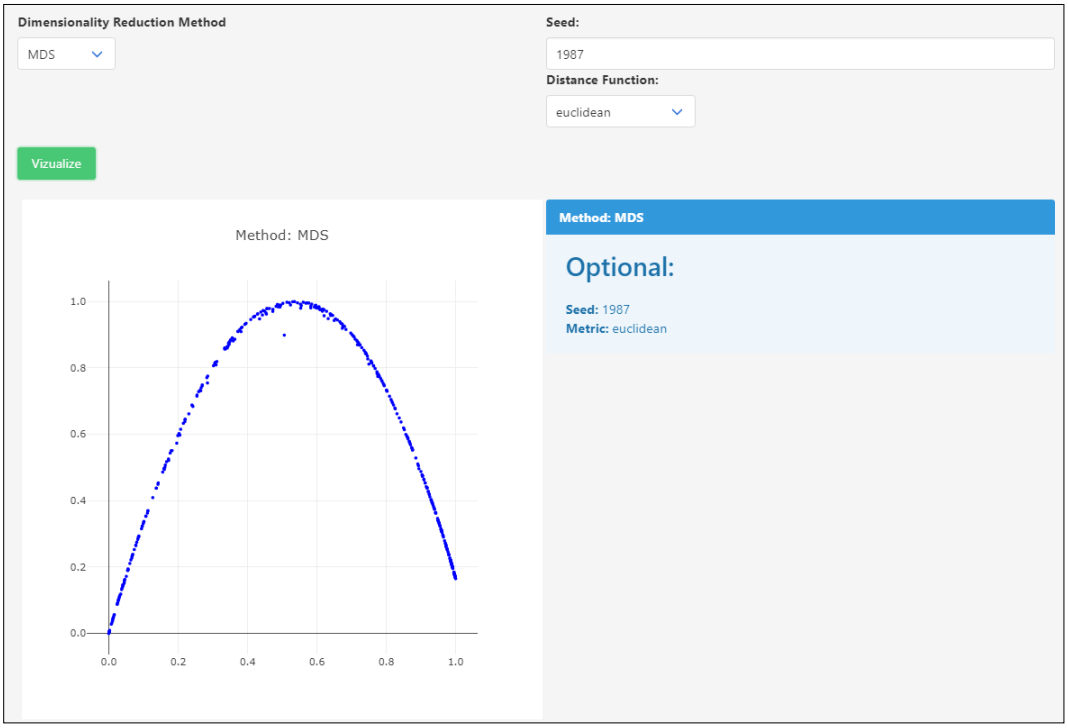

Figure 3.8: Graph produced with MDS.

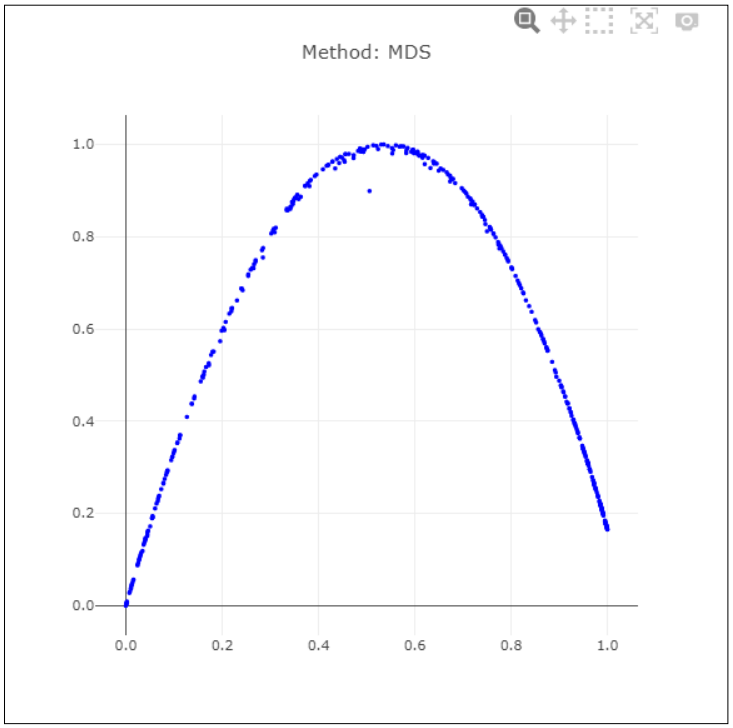

Figure 3.9: Five options are provided as iconic buttons at the top right of the graph window.

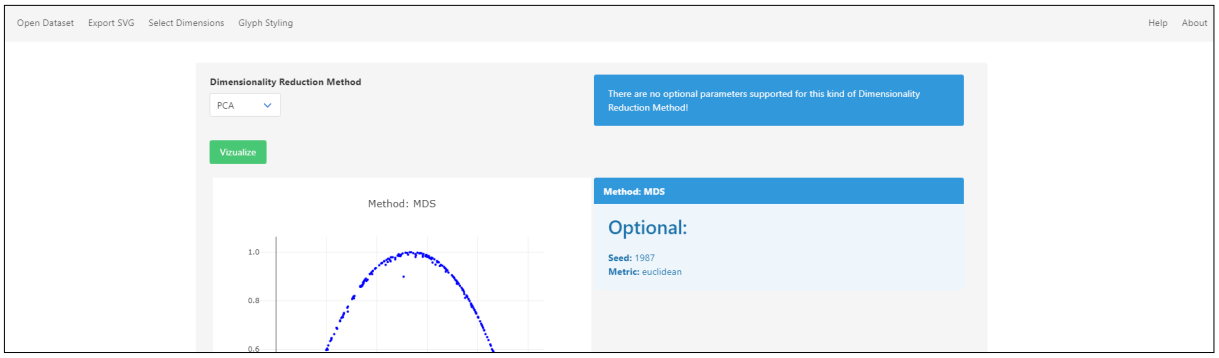

Figure 3.10: After a chart has been created, dimensions can be added or removed with the Select Dimensions button.

Figure 3.2.

3.5 Glyph Styling

On the navigation bar shown in Figure 3.10, the user can click the Glyph Styling button to open the Glyph Styling pop-up window shown in Figure 3.11. This allows the user to select the shape, size, and color for the plotted gylphs. Clicking the Visualize button once again will apply the chosen settings, leading to Figure 3.12.

3.6 Help and About

At the top right of the application, there are Help and About buttons, as shown in Figure 3.10. If the user clicks on the Help button, they will be redirected to the help page shown in Figure 3.13, which explains

Figure 3.11: Pop-up window for glyph styling.

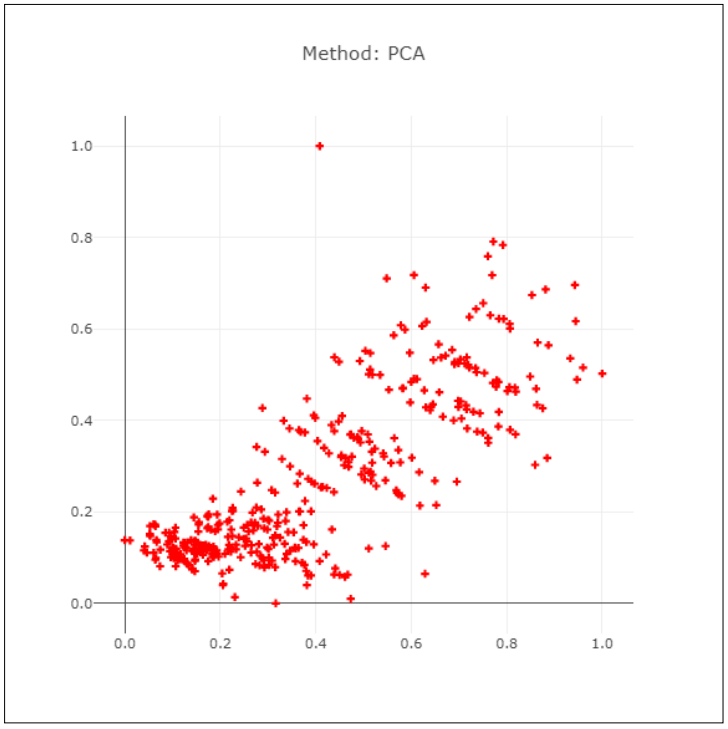

Figure 3.12: Graph with new glyphs.

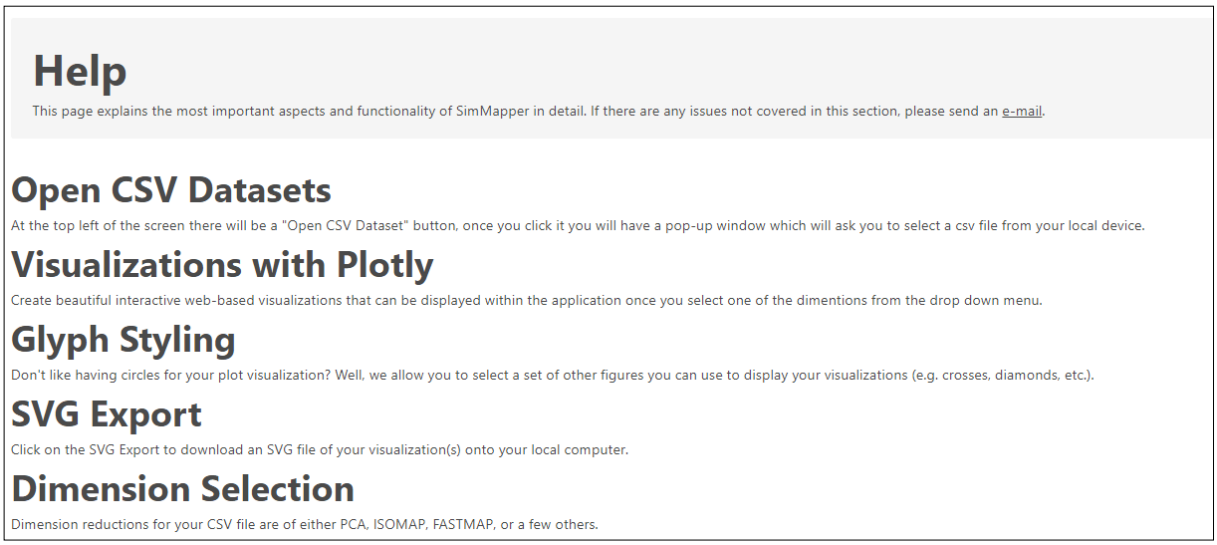

Figure 3.13: The Help page.

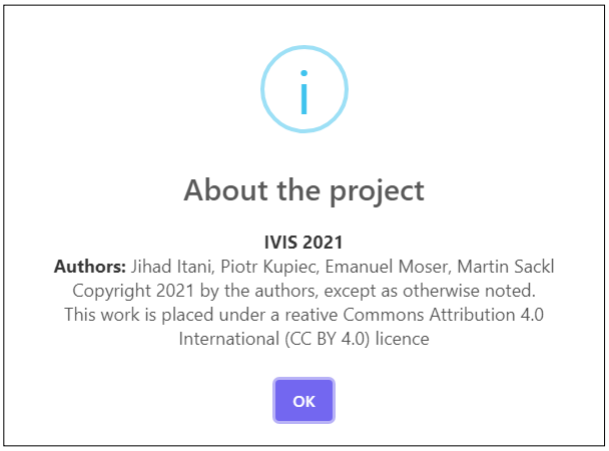

Figure 3.14: The About window.

the most important aspects and functionality of SimMapper in detail. Finally, the About button display a small pop-up window with background information about the project, as shown in Figure 3.14.

Chapter 4

Future Work

In this chapter, we want to showcase some ideas we had during the development of SimMapper, but were unable to realize due to lack of time or their being out of the scope of our current project.

4.1 Functionality

In this section, we suggest some potential future improvements and extensions to the functionality of SimMapper.

4.1.1 Multiple Visualization View

As can be seen in Figure 4.1, we already had a working implementation showing multiple graphs in a grid pattern, which was also semi-responsive. We went back to the single graph display for the time being for simplicity. We propose a view with a collapsed details dialog e.g. beneath every visualization so a user can visually compare different results. Should more detailed information be desired, the collapsed detail panel can be opened up to reveal the specific parameters per visualization.

4.1.2 Individual Graph Update

In addition, it would be beneficial for a user to be able to automatically adjust the parameters of a given graph (type of reduction, metric, etc). The details dialogue could be refactored to function as a form field for the specified graph. This should not be done lightly, because it requires a complete refactoring of our existing code.

4.1.3 More Visualization Options

Currently, it is possible to choose among a small collection of glyphs which can be customized by size and color. In the future it would be beneficial to allow a user to add custom glyphs. We currently only have a choice of 7 colors for the glyphs. We propose to implement a color-picker so a user can change it more easily. Furthermore a small preview of the glyph would be interesting, just to see how it looks before it gets used in an visualization.

Once these changes are implemented, it might be beneficial for a user to have custom glyph presets that persist to a later session of SimMapper. This might be achievable by persisting the information about glyphs into a configuration .json file which would also allow sharing it with other users.

4.1.4 3D Charts

In its current version, SimMapper reduces every dataset to 2 dimensions and renders them in a 2-dimensional scatterplot. Since DruidJS does offer the opportunity to reduce to more than 2 dimensions, we think it would also be interesting to show a 3-dimensional scatterplot.

Figure 4.1: Potential display for multiple visualizations.

4.1.5 Optimize Export Functionality

As described in Section 2.3, SimMapper has export functionality to SVG, which creates a responsive SVG file. Nevertheless, this file is still not easily human-readable: there are no line breaks, real numbers are given with overly many digits of precision, etc. Some (optional) post-processing after generation might be beneficial.

4.2 Usability Improvements

For general usability improvements, we have the following suggestions.

4.2.1 Drag-and-Drop

Currently, we only allow for manually clicking on Open Dataset, which in turn opens the system dialogue to select a file. When looking at Figure 3.1, and in the future it should be possible to drag a CSV file directly into SimMapper or directly onto the Open Dataset button which automatically triggers SimMapper to open the CSV file, followed by the preview dialogue.

4.2.2 Svelte Components

Some components in SimMapper are used repeatedly, such as the details page of a visualization or the snippet rendering the CSV preview upon opening a file. We propose building Svelte [Svelte 2021] components, which have the advantage of being placeholders, being independent from the rest of our code, and being readily re-usable. One example would be to put the visualization and details into such a component and just render a new one or replace an old component. If a multi-visualization view as described in Subsection 4.1.1 is implemented, it might be beneficial to implement Svelte components beforehand.

Bibliography

Bostock, Mike [2021]. *D3: Data-Driven Documents*. 01 Jul 2021. https://d3js.org/ (cited on page 2).

Cutura, Rene, Christoph Kralj, and Michael Sedlmair [2020]. *DRUID JS – A JavaScript Library for Dimensionality Reduction*. Proc. IEEE Visualization Conference (Vis 2020) (Virtual). 25 Oct 2020, pages 111–115. doi:10.1109/VIS47514.2020.00029. https://renecutura.eu/pdfs/Druid.pdf (cited on page 1).

Cutura, Rene and Philippe Rivière [2021]. *DruidJS*. 01 Jul 2021. https://github.com/saehm/DruidJS (cited on pages 1–2).

Gulp [2021]. *Gulp*. 01 Jul 2021. https://gulpjs.com/ (cited on page 2).

Itani, Jihad, Piotr Kupiec, Emanuel Moser, and Martin Sackl [2021a]. *SimMapper GitLab Page*. 02 Jul 2021. https://mj_massacre.gitlab.io/simmapper/view/ (cited on page 1).

Itani, Jihad, Piotr Kupiec, Emanuel Moser, and Martin Sackl [2021b]. *SimMapper GitLab Repository*. 02 Jul 2021. https://gitlab.com/mj_massacre/simmapper (cited on page 1).

Microsoft [2017]. *TypeScript: Typed JavaScript at Any Scale*. 01 Jun 2017. https://typescriptlang.org/ (cited on page 2).

OpenJS [2017]. *Electron*. OpenJS Foundation, 01 Jun 2017. https://electronjs.org/ (cited on page 1).

Plotly [2021]. *plotly.js*. 01 Jul 2021. https://plotly.com/javascript/ (cited on page 2).

Pramoditha, Rukshan [2021]. *11 Dimensionality Reduction Techniques You Should Know in 2021*. 14 Apr 2021. https://towardsdatascience.com/11-dimensionality-reduction-techniques-you-should-know-in-2021-dcb9500d388b (cited on page 1).

Sackl, Martin [2021]. *SimMapper Showcase Video*. 29 Jun 2021. https://youtu.be/fHhMxltjGOw (cited on page 7).

Sikelianos, Zeke [2017]. *Announcing TypeScript Support in Electron*. 01 Jun 2017. https://electronjs.org/blog/typescript (cited on page 1).

Stackgl [2020]. *Modular WebGL Components*. 07 Jun 2020. https://github.com/stackgl (cited on page 2).

Svelte [2021]. *Svelte*. 01 Jul 2021. https://svelte.dev/ (cited on page 16).

Thomas, Jeremy [2021]. *Bulma*. 01 Jul 2021. https://bulma.io/ (cited on page 2).