Assignment 5: Parallel Coordinates

Date Posted: March 15, 2024

Due Date: April 5, 2024 11:59pm

Description

In this assignment, you will implement interactive parallel coordinates. This time, we will use the popular iris dataset.

Submission Requirements

Provided Template

In the GitHub repository, you will find six files: `index.html`, `plot.js`, `iris.js`, `d3.v7.min.js`, `README.md`, and `spec.pdf`. The `index.html` provides a minimal HTML outline to set up your webpage and structure your responses. The `plot.js` file contains a template for your JavaScript code and is where you should write your plotting code. The `iris.js` file contains the data for your parallel coordinates. `d3.v7.min.js` is a copy of the d3 library. You will need to load this before any other scripts that use d3. `README.md` provides a template to use for documenting your repository as described below. Finally, `spec.pdf` contains a copy of the assignment specification (this document).

Submitted Files

You will create your visualizations in `index.html`, it should be the only HTML file in your submission. You should write your main plotting code in `plot.js`, but may add any additional JavaScript and CSS files as you see fit. You must add appropriate documentation via commenting to all JavaScript code. Finally, you should include a `README.md` file (using the given template) that provides a text description of what is in your repository, how to run your program and any parameters that you used. Also, document any idiosyncrasies, behaviors, or bugs of note that you want us to be aware of. **When updating the README, please remove/update the instructive text in parentheses - that is only there to guide you**. When you have completed your assignment, please submit the link to your GitHub repository on Canvas.

You must use `d3.js` to create your visualizations. However, you may also use any other external libraries that you see fit. If you opt to use additional libraries, please document which libraries and how you used them in your `README.md` file.

Creating Parallel Coordinates in D3

In this assignment, you will implement a visualization using parallel coordinates to display the Iris dataset. Each item in this dataset represents an Iris flower. Each flower is described by 5 attributes - sepalWidth,

sepalLength, petalWidth, petalLength and species. The species is a categorical attribute with 3 classes - Setosa, Versicolor, and Virginica. The other 4 attributes are quantitative. The data contains 150 items, 50 in each species type.

Your parallel coordinates plot should draw all four dimensions simultaneously (on individual **vertical** axes) and allow people to interactively inspect how items span the four attributes. Specifically, you should allow people to (1) brush on each axis, highlighting the items that are selected in **all** active brushes, and (2) re-order the axes.

In this assignment, you will be using two types of data bindings - ones that map the actual iris data (called `data` in the template) to the elements and ones that map an array of the quantitative data attributes (called `attrs` in the template) to elements. You will create several bindings using the array of attributes to create our brushable, re-orderable axes.

In the template, I have provided some initialized objects to help define your axes - `axes`,`yScales`,`brushes` and `brushBounds`. These will be described in more detail before. You can modify these as you see fit but they should be sufficient for setting up your plot.

There are three steps needed to complete this assignment, detailed below. You do not necessarily have to complete them in this order.

Step 1: Drawing the Marks

In this step, you will create line marks for each item in the dataset using the SVG `path` element. These should be drawn as a polyline with four points, one point per quantitative attribute (stored in `attrs`). To do this, you will use `d3.line()`, which accepts an array of coordinate pairs and returns the associated SVG path attribute `d`. For example, `d3.line()([[x0,y0],[x1,y1],...])` creates a line that starts at [x0,y0], then moves to [x1,y1], and so on. You will need to use the pre-defined y-scales to compute the coordinates for each axis.

I have provided both `xScale` and `yScales` to help do this. `xScale` is of type `d3.scalePoint()` ad will return the x-position for any attribute in `attrs`. `yScales` is an object that contains all of the y-scales. There is a key for each attribute in `attrs` that maps to the corresponding y-Scale. For example, `yScales[attr[0]]` returns the y-scale for the first attribute.

You can find documentation on how to use `d3.line()` here.

Additionally, you should color the lines by the `species` attribute of each data item, and the opacity should be 0.75. You should set the class of each path to `dataline`.

Step 2: Drawing the Axes

To draw the axes, you will need to perform 3 selections, binding to the `attrs`.

First, you will need to create a selection of group elements and bind them to `attrs`. These groups are where you will add your axes. You will need to appropriately transform each group element (to draw it in the correct position), as well as call the corresponding axis stored in the `axes` object.

Recall, normally we use `axis_group.call(axis)` to display an axis. However, because we have a selection of multiple groups, we will not use `.call()` on the whole selection. Instead, we will use d3's `.each()` function to iterate over the selection and call the axis on each group individually. It should look something like this: `.each(function(d){ return d3.select(this).call(axes[d]);})`

Next, you will need to create a selection of group elements and bind them to `attrs`. These groups are where you will add your axis labels. You will need to appropriately transform each group element (to draw it in the correct position, **centered above** the corresponding axis). These elements should have the class `label`. You will need to add a click event handler to each label (using `.on()`) to support a click interaction (described in the next step).

Finally, for each axis we will create a group for each brush. These groups will have the class `brush`. Similar to the first selection, you will need to use the `.each()` function to call the appropriate brush on the group. Additionally, you will need to transform the brush to be on the correct axis.

Step 3: Interaction

You will implement two interactions: mouse click and brushing.

Mouse Click

First, when a text label is clicked, you must swap the axis with the the one to its right. If the rightmost label is clicked, it should swap with the one on its left.

To do this, you should swap the attributes in the `attrs` array. After doing this, you will need to update everything that depends on the `attrs` array (e.g. the paths, the xScale, the axis bindings, etc.).

Brushing

he second interaction should enable people to brush over a each individual axes to select a subset of the data. If multiple axes are brushed, the selection must fall within all of the brush bounds.

You will implement this similar to HW 4. You will fill in the function `isSelected()` to return true for an data items that is within the bounds of **all** of the active brushes. The brushes we created are 1-dimensional, using `d3.brushY()`, which means that their bounds are a bit simpler. Each brush selection only contains two elements [y0,y1] where y0 is the minimum of the brush and y1 is the maximum. For a given attribute, brushBounds[attr] contains [y0,y1] (in pixels).

Selected lines should have an opacity of .75, while un-selected lines should have an opacity of 0.1. If no brushes are active, then all elements should have opacity .75

Grading

Your submission will be graded on how well it fulfills the specified criteria. Additionally, 15 points will be deducted for submissions that do not build (i.e. whose plots do not show up). The below rubric describes the point values for individual components.

Implementing the line marks	20
Correctly adding the groups for axes, labels and brushes.	20
Implementing the click interaction	25
Correctyl selecting and updating opacity using brushes	25
Proper documentation (code comments and README.md)	10