

Enhanced Card Sorting Analysis in R

Group 5:

Michaela Kargl, Ajdin Mehic, Zoran Prodanovic, and David Seywald

706.057 Information Visualisation SS 2018
Graz University of Technology

27 June 2018

Abstract

This project report describes the work carried out to enhance and improve the prototype card sorting analysis tool CSA. First, a brief introduction to the method of card sorting is given. Then, all functionalities of the improved CSA prototype are described and explained in detail. The main features of the improved CSA prototype include absolute and normalised versions of the similarity and distance matrices, a coloured version of the co-occurrence matrix, similarity maps created with 3 different algorithms, a dendrogram for visualisation of the results in a tree format, and the representation of the result-groups in tabular format together with a suggestion of the most frequently assigned group labels.

© Copyright 2018 by the author(s), except as otherwise noted.

This work is placed under a Creative Commons Attribution 4.0 International (CC BY 4.0) licence.

Contents

- Contents** **i**

- List of Figures** **iii**

- List of Tables** **v**

- List of Listings** **vii**

- 1 Introduction** **1**

- 2 About the Card Sorting Analysis Application CSA** **3**

- 3 Data Input** **5**

- 4 Similarity Matrix** **7**

- 5 Distance Matrix** **11**

- 6 Distance Histogram** **15**

- 7 Co-Occurrence Matrix** **19**

- 8 Similarity Maps** **21**
 - 8.1 Similarity Map with t-SNE 22
 - 8.2 Similarity Map with MDS 24
 - 8.3 Similarity Map with FDP 26

- 9 Dendrogram** **29**

- 10 Grouped Results** **31**

- 11 Conclusion** **33**

- Bibliography** **35**

List of Figures

2.1	CSA Readme Tab Information	4
3.1	CSA Data Tab used for data upload	6
4.1	Screenshot of the Tab "Similarity" of the CSA Application	8
4.2	[Screenshot of the Tab "Similarity" of the CSA Application	9
5.1	Screenshot of the Tab "Distance" of the CSA Application	12
5.2	[Screenshot of the Tab "Distance" of the CSA Application	13
6.1	Screenshot of the Tab "Histogram" of the CSA Application	16
7.1	SVG Legend for Co-Occurrence Matrix	20
7.2	Screenshot of the Tab "Co-Occurrence Matrix" of the CSA Application	20
8.1	Screenshot of the Tab "SimilarityMap" of the CSA Application	22
8.2	Similarity Map Created by the CSA Application with a t-SNE Algorithm	23
8.3	Parameters used for the t-SNE Algorithm	24
8.4	Similarity Map Created by the CSA Application with a MDS Algorithm	26
8.5	Similarity Map with Force-directed Placement Created by the CSA Application	28
9.1	Screenshot of the Tab "Dendrogram" of the CSA Application	30
10.1	Screenshot of the Tab "Results" of the CSA Application	32

List of Tables

List of Listings

6.1	Code for Plotting the Histogram in the CSA Application	17
7.1	Shiny JavaScript Output Event	20
8.1	Code for Plotting the Similarity Map with t-SNE Algorithm in the CSA Application	23
8.2	Code for Plotting the Similarity Map with MDS Algorithm in the CSA Application	25
8.3	Code for Plotting the Similarity Map with FDP in the CSA Application	27
10.1	Result Group Names Calculation	32

Chapter 1

Introduction

The Card Sorting Analysis Tool is based on the previous work:

"Card Sorting Analysis with Spreadsheet and R" by Ines Terzic, Lukas Krisper, Stefan Painhapp, and Manuel Papst from 2016.

An Introduction and Overview about Card Sorting can be found in their paper. The main focus of our work was to fix some minor bugs in the already existing software as well as implementing some new features.

Chapter 2

About the Card Sorting Analysis Application CSA

The Card Sorting Analysis Application (CSA) is based on previous work from 2016. It is a Shiny web application written in R. For easy access via browser it is hosted on shinyapps.io under the following url: <https://infovis-tug.shinyapps.io/csa18/>

This work is licensed under the MIT license.

The CSA is divided into multiple Tabs:

- Readme - Start page, contains information/license text
- Data - First page, here the csv data is uploaded
- Similarity - Contains the (normalised) similarity matrix
- Distance - Contains the (normalised) distance matrix
- Histogram - Improved Histogram implementation
- Co-Occurrence - Color coded Co-Occurrence matrix
- Similarity Map - Different algorithms (t-SNE, MDS, FDP)
- Dendrogram - Different clustering algorithms with graphical output
- Result - Grouped results, with meaningful group name calculation

In the following chapters the functionality of the CSA card sorting analysis application are described in more detail. Figure 2.1 shows a screenshot of current implementation of the Readme tab.

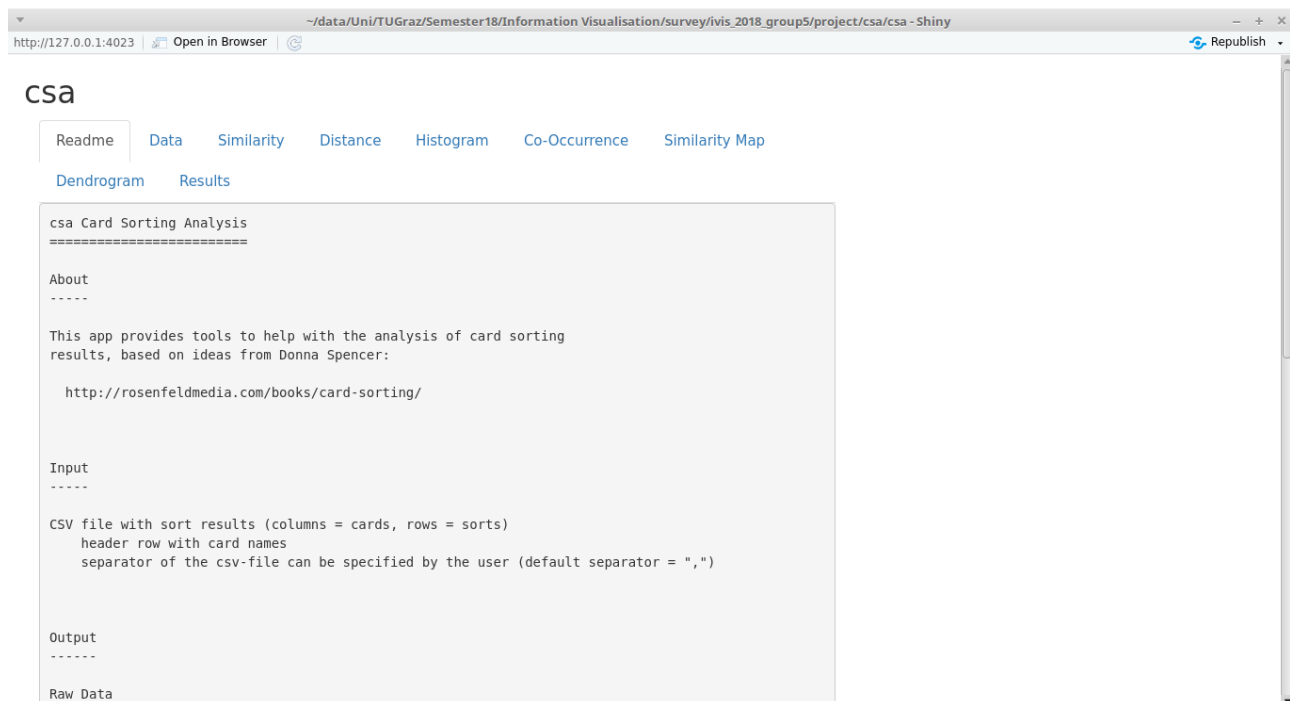


Figure 2.1: CSA Readme Tab Information/License

Chapter 3

Data Input

The tab "Data" of the CSA application is the first tab which needs to be entered to be able to use the CSA tool in a meaningful way. First of all the user has to define which character (comma, semicolon etc.) is being used as separator in the input file, which should be a CSV. The separator is used for reading the input file as well as writing output later on, e.g. when a matrix is being downloaded. Right after that there is a button where the user has to select which CSV should be used as input file. After the file upload finishes we also display some information about the input file to the user: the file name, the number of rows as well as the number of columns. Furthermore the data is immediately displayed to the screen, so that the user can check if the correct data set was selected. Figure 3.1 shows a screenshot of the Data tab of the CSA application after a file has been successfully uploaded.

csa

Readme **Data** Similarity Distance Histogram Co-Occurrence Similarity Map Dendrogram Results

Input

CSV Separator

Choose file to upload

Filename: mindset1.csv
Rows: 38
Columns: 100

	Sausages	Beer	Parsley	Mustard	Hand Cream	Teabags	After Shave	Honey	Whipping Cream	Bisc
1	Meat	Drinks	Vegetables	Sauces & Spices	Hygiene	Staple Food	Hygiene	Sweets	Sweets	Swe
2	Meat	Alcoholic Drinks	Sauces & Spices	Sauces & Spices	Hygiene	Non-Alcoholic Drinks	Hygiene	Sauces & Spices	Milk Products	Swe
3	Meat	Alcoholic Drinks	Vegetables	Sauces & Spices	Hygiene	Coffee & Tea	Hygiene	Breakfast	Fridge	Swe
4	Meat	Alcoholic Drinks	Sauces & Spices	Dressings	Hygiene	Non-Alcoholic Drinks	Hygiene	Breakfast	Milk Products	Swe
5	Meat	Alcoholic Drinks	Snacks	Snacks	Hygiene	Non-Alcoholic Drinks	Hygiene	Breakfast	Milk Products	Swe
6	Meat	Alcoholic Drinks	Sauces & Spices	Dressings	Hygiene	Snacks	Hygiene	Breakfast	Milk Products	Baki
7	Meat	Drinks	Vegetables	Sauces & Spices	Hygiene	Household	Hygiene	Breakfast	Milk Products	Sna
8	Convenience Products	Alcoholic Drinks	Vegetables	Staple Food	Hygiene	Breakfast	Hygiene	Staple Food	Staple Food	Swe
9	Meat	Drinks	Venetables	Sauces &	Hvniene	Drinks	Hvniene	Sauces &	Sweets	Swe

Figure 3.1: CSA Data Tab used for data upload

Chapter 4

Similarity Matrix


The tab "Similarity" of the CSA application shows the calculated similarity matrix of the input data. The values in the similarity matrix state how often each row-item was classified into the same category with the corresponding column-item. The result is a lower triangular matrix, since the values are mirrored along the main diagonal.

Furthermore it consists of a checkbox to enable the user to switch between a similarity matrix with absolute and one with normalised values. Additionally we provide a download-button for saving the matrix as a CSV, where the given value from the data tab is used as separator value. Figure 4.1 shows a screenshot of the "Similarity" tab with absolute values and Figure 4.2 is the same table with normalised values.

CSA

[Readme](#)
[Data](#)
[Similarity](#)
[Distance](#)
[Histogram](#)
[Co-Occurrence](#)
[Similarity Map](#)
[Dendrogram](#)
[Results](#)

normalised


[Download Similarity Matrix](#)

	Sausages	Beer	Parsley	Mustard	Hand Cream	Teabags	After Shave	Honey	Whipping Cream	Biscuits
Sausages	38									
Beer	1	38								
Parsley	3	0	38							
Mustard	3	0	16	38						
Hand Cream	0	0	0	0	38					
Teabags	4	9	2	4	2	38				
After Shave	0	0	0	0	38	2	38			
Honey	3	0	10	14	0	10	0	38		
Whipping Cream	6	0	4	3	0	2	0	9	38	
Biscuits	2	0	1	1	0	4	0	13	7	38

Figure 4.1: The tab "Similarity" of the CSA application shows the calculated similarity matrix of the input data with absolute values.

CSA

- Readme
- Data
- Similarity
- Distance
- Histogram
- Co-Occurrence
- Similarity Map
- Dendrogram
- Results

normalised

Download Normalised Similarity Matrix

	Sausages	Beer	Parsley	Mustard	Hand Cream	Teabags	After Shave	Honey	Whipping Cream	Biscuits
Sausages	1									
Beer	0.03	1								
Parsley	0.08	0	1							
Mustard	0.08	0	0.42	1						
Hand Cream	0	0	0	0	1					
Teabags	0.11	0.24	0.05	0.11	0.05	1				
After Shave	0	0	0	0	1	0.05	1			
Honey	0.08	0	0.26	0.37	0	0.26	0	1		
Whipping Cream	0.16	0	0.11	0.08	0	0.05	0	0.24	1	
Biscuits	0.05	0	0.03	0.03	0	0.11	0	0.34	0.18	1
Strawberries	0.03	0	0.03	0.03	0	0	0	0	0.05	0
Walnuts	0.03	0	0.05	0.05	0	0.05	0	0.18	0.18	0.26

Figure 4.2: The tab "Similarity" of the CSA application shows the calculated similarity matrix of the input data with normalised values.

Chapter 5

Distance Matrix


The tab "Distance" of the CSA application shows the calculated distance matrix of the input data. The values in the distance matrix state how often each row-item was classified into a different category than the corresponding column-item. The result is a lower triangular matrix, since the values are mirrored along the main diagonal.

Furthermore it consists of a checkbox to enable the user to switch between a distance matrix with absolute and one with normalised values. Additionally we provide a download-button for saving the matrix as a CSV, where the given value from the data tab is used as separator value. Figure 5.1 shows a screenshot of the "Distance" tab with absolute values and Figure 5.2 is the same table with normalised values.

CSA

Readme Data Similarity **Distance** Histogram Co-Occurrence Similarity Map Dendrogram Results

normalised

 Download Distance Matrix


	Sausages	Beer	Parsley	Mustard	Hand Cream	Teabags	After Shave	Honey	Whipping Cream	Biscuits	Strawberries	Walnuts
Sausages	0											
Beer	37	0										
Parsley	35	38	0									
Mustard	35	38	22	0								
Hand Cream	38	38	38	38	0							
Teabags	34	29	36	34	36	0						
After Shave	38	38	38	38	0	36	0					
Honey	35	38	28	24	38	28	38	0				
Whipping Cream	32	38	34	35	38	36	38	29	0			
Biscuits	36	38	37	37	38	34	38	25	31	0		
Strawberries	37	38	37	37	38	38	38	38	36	38	0	
Walnuts	37	38	36	36	38	35	38	31	31	28	38	0
Cucumbers	36	38	15	35	38	37	38	37	36	38	33	37
Beef	3	37	35	33	38	34	38	34	33	37	37	37

Figure 5.1: The tab "Distance" of the CSA application shows the calculated distance matrix of the input data with absolute values.

CSA

Readme Data Similarity **Distance** Histogram Co-Occurrence Similarity Map Dendrogram Results

normalised

 Download Normalised Distance Matrix

	Sausages	Beer	Parsley	Mustard	Hand Cream	Teabags	After Shave	Honey	Whipping Cream	Biscuits	Strawberries	Walnuts
Sausages	0											
Beer	0.97	0										
Parsley	0.92	1	0									
Mustard	0.92	1	0.58	0								
Hand Cream	1	1	1	1	0							
Teabags	0.89	0.76	0.95	0.89	0.95	0						
After Shave	1	1	1	1	0	0.95	0					
Honey	0.92	1	0.74	0.63	1	0.74	1	0				
Whipping Cream	0.84	1	0.89	0.92	1	0.95	1	0.76	0			
Biscuits	0.95	1	0.97	0.97	1	0.89	1	0.66	0.82	0		
Strawberries	0.97	1	0.97	0.97	1	1	1	1	0.95	1	0	
Walnuts	0.97	1	0.95	0.95	1	0.92	1	0.82	0.82	0.74	1	0
Cucumbers	0.95	1	0.39	0.92	1	0.97	1	0.97	0.95	1	0.87	0.97
Beef	0.08	0.97	0.92	0.87	1	0.89	1	0.89	0.87	0.97	0.97	0.97

Figure 5.2: The tab "Distance" of the CSA application shows the calculated distance matrix of the input data with normalised values.

Chapter 6

Distance Histogram

The tab "Histogram" of the CSA application shows a histogram of the content of the distance matrix, and provides two download-buttons for saving the histogram graphic as png-file and as svg-file respectively. Figure 6.1 shows a screenshot of the tab "Histogram" of the CSA application.

The histogram is plotted using the function `hist()` from the base R graphics package. A detailed description of the capabilities and usage of `hist()` can be found in the R Documentation [Team 2015a]. The code snippet in Listing 6.1 shows how the `hist()` function is utilised in the `csa` application for plotting the histogram.

The breakpoints between the histogram cells are set in such a way that each histogram cell includes one value from the range of possible values in the distance matrix. Since the values in the distance matrix show for each pair of cards the number of sorts which had assigned these two cards to different groups, the lowest possible value in the distance matrix is zero, and the highest possible value in the distance matrix is equal to the number of sorts. The heights of the bars of the histogram show for each of the values how often that specific value appeared in the distance matrix. The maximum value shown on the y-axis equals the total number of cells in the distance matrix, as this is the maximum possible count.

Such a histogram graphic is useful to get a quick overview regarding the variety of the sorting results. If there are high bars at the edges of the histogram and almost no counts in middle bins of the histogram, this is an indication for a high degree of congruence among the results of the single sorts. Such a high degree of congruence among the results of the single sorts facilitates the final grouping. For example, from the histogram shown in Figure 6.1 it can be seen that 4768 of the 10000 possible card-pairs were split and sorted into different groups by all of the 38 test-users. Thus, for about 47 percent of the card pairs it seems quite clear that the elements of that card-pairs should not be together in a group in the final grouping-version. Furthermore, from the histogram in Figure 6.1 it can also be seen that 352 card-pairs were split into different groups by none of the test-users. Thus it seems quite clear that the elements of these 352 card pairs should also be sorted into the same group in the final grouping-version.

csa

[Readme](#) [Data](#) [Similarity](#) [Distance](#) **[Histogram](#)** [Co-Occurrence](#) [Similarity Map](#)
[Dendrogram](#) [Results](#)

[Download Histogram as png](#)

[Download Histogram as svg](#)

Distance Histogram

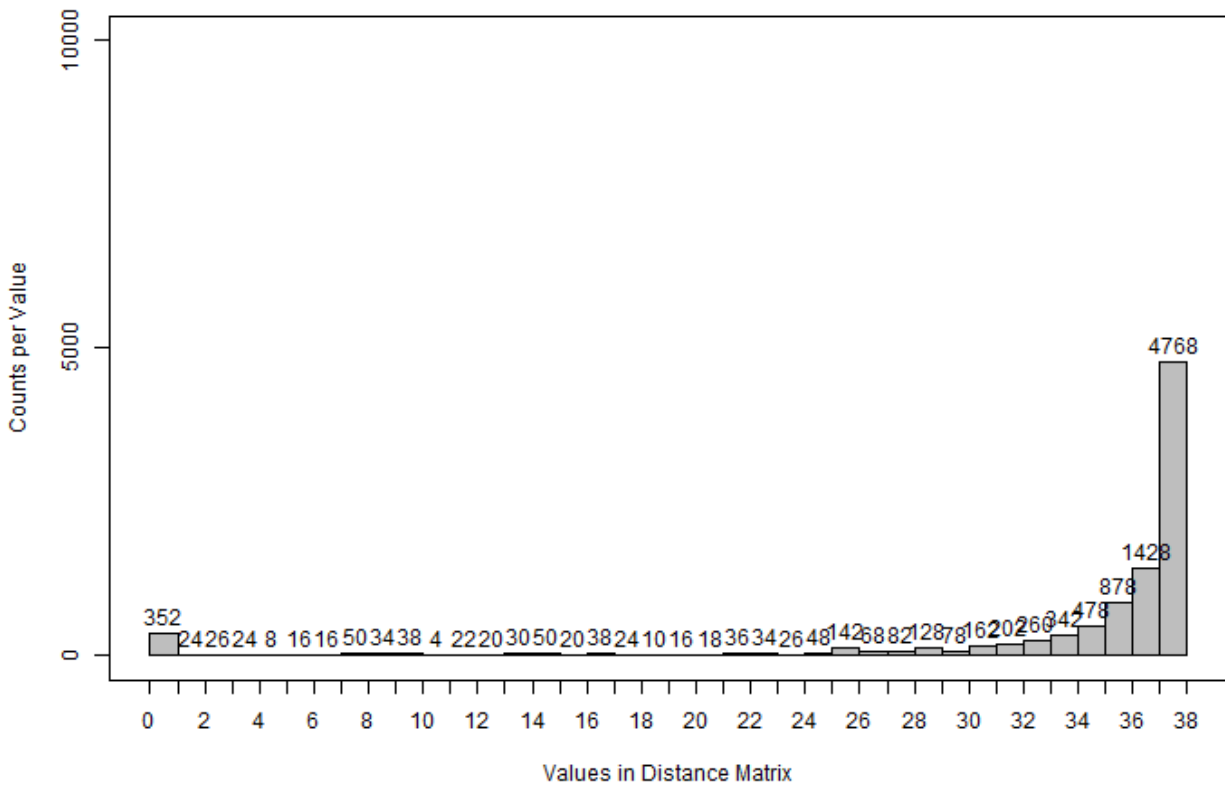


Figure 6.1: The tab "Histogram" of the CSA application shows a histogram of the content of the distance matrix, and provides the possibility to download this graphic as png-file or svg-file.

```

1 plotHisto = function() {
2   # draw the histogram of the distance matrix
3   hst <- hist(csdistance(), col = "gray",
4             # add a diagram title and axis labels
5             main = "Distance Histogram",
6             xlab = "Values in Distance Matrix",
7             ylab = "Counts per Value",
8             # the max value on the y axis equals the size of the distance matrix
9             ylim = c(0, (ncol(csdistance()) * ncol(csdistance()))),
10            # display the labels of the data
11            labels = TRUE,
12            # set number of histogram-bins to range of distance matrix values
13            breaks = c(0:nrOfLines()),
14            # upper and lower limits of the range are included
15            right = TRUE, include.lowest = TRUE,
16            # do not display the default axes
17            axes = FALSE
18          )
19   # add custom x-axis
20   axis(1, c(0:nrOfLines()))
21   # add custom y-axis
22   axis(2, at = c(0,(ncol(csdistance()) * ncol(csdistance()))/2,
23                (ncol(csdistance()) * ncol(csdistance()))
24          )
25   # draw a box around the graphic
26   box()
27   return(hst)
28 }

```

Listing 6.1: The code for plotting the histogram in the CSA application utilises the `hist()` function from the base R graphics package and adds custom axes and a box.

Chapter 7

Co-Occurrence Matrix

The tab "Co-Occurrence Matrix" of the CSA application shows the Co-Occurrence Matrix. All the values are given in percent and are color coded based on their range from white to dark blue. New added functionality includes:

- Download Button (csv)
- Color coded matrix (via JavaScript)
- SVG Legend explaining the values/color ranges

The table is rendered as usual, but in the background an included JavaScript file is listening on the built-in shiny js events, and triggering the color coding functions after the Co-Occurrence Matrix table is drawn.

```

1 $(document).on("shiny:value", function(e) {
2   if (e.name == "cooccurrenceMatrix") { //the id of the output element
3     e.preventDefault();
4     $("#cooccurrenceMatrix")
5       .removeClass("shiny-output-error") // get rid of potential previous error styling
6       .html(e.value); // render the output from the server
7     colorCode();
8   }
9 });

```

Listing 7.1: JavaScript code for listening on built-in shiny js events to color code the Co-Occurrence Matrix.

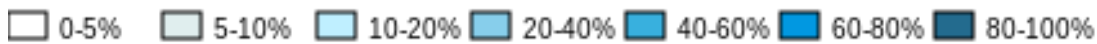


Figure 7.1: SVG Legend for Co-Occurrence Matrix

	Sausages	Beer	Parsley	Mustard	Hand Cream	Teabags	After Shave	Honey	Whipping Cream	Biscuits	Strawberries	Walnuts	Cucum
Sausages	100												
Beer	2.63	100											
Parsley	7.89	0	100										
Mustard	7.89	0	42.11	100									
Hand Cream	0	0	0	0	100								
Teabags	10.53	23.68	5.26	10.53	5.26	100							
After Shave	0	0	0	0	100	5.26	100						
Honey	7.89	0	26.32	36.84	0	26.32	0	100					
Whipping Cream	15.79	0	10.53	7.89	0	5.26	0	23.68	100				
Biscuits	5.26	0	2.63	2.63	0	10.53	0	34.21	18.42	100			
Strawberries	2.63	0	2.63	2.63	0	0	0	0	5.26	0	100		
Walnuts	2.63	0	5.26	5.26	0	7.89	0	18.42	18.42	26.32	0	100	
Cucumbers	5.26	0	60.53	7.89	0	2.63	0	2.63	5.26	0	13.16	2.63	100
Beef	92.11	2.63	7.89	13.16	0	10.53	0	10.53	13.16	2.63	2.63	2.63	100
Kitchen Roll	0	0	0	0	31.58	5.26	31.58	0	0	0	0	0	0
Apple Juice	5.26	31.58	2.63	2.63	0	39.47	0	0	2.63	0	2.63	0	0
Red Wine	2.63	100	0	0	0	23.68	0	0	0	0	0	0	0
Frozen Pizza	15.79	0	10.53	7.89	0	7.89	0	5.26	10.53	2.63	2.63	2.63	100
Body Lotion	0	0	0	0	100	5.26	100	0	0	0	0	0	0

Figure 7.2: The tab "Co-Occurrence Matrix" of the CSA application shows the Co-Occurrence Matrix, and provides the possibility to download this data as csv-file.

Chapter 8

Similarity Maps

Similarity maps are a visualisation technique for high-dimensional data, aiming at making similarities between elements visible. The aim is to achieve similarity maps, where "similar" elements are displayed close together and groups can easily be distinguished visually. The similarity map provided in the CSA application is a projection of the high-dimensional similarity matrix into the two-dimensional space, in order to provide a visual clue for grouping of the elements.

As depicted in Figure 8.1, within the tab "Similarity Map" of the CSA application the user can choose among 3 different algorithms for the creation of a similarity map: MDS (Multi-Dimensional Scaling), FDP (Force-Directed Placement), and t-SNE (t-Distributed Stochastic Neighbor Embedding). By default a similarity map created by the t-SNE algorithm is displayed. On top of the tab two download-buttons are available to save the similarity map graphic as png-file or svg-file respectively.

In the following sections the similarity maps based on the MDS, FDP, and t-SNE algorithms are described in more detail.

CSA

- Readme
- Data
- Similarity
- Distance
- Histogram
- Co-Occurrence
- Similarity Map
- Dendrogram

Results

Layout Algorithm:

MDS

MDS

FDP

t-SNE

Download Similarity Map as png

Download Similarity Map as svg

Similarity Map (MDS)

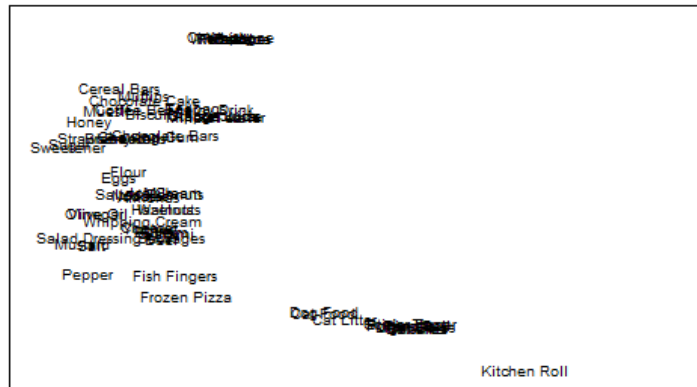


Figure 8.1: The tab "Similarity Map" shows a similarity map based on the user's algorithm-choice, and provides the possibility to download this graphic as png-file or svg-file.

8.1 Similarity Map with t-SNE

t-SNE is an abbreviation and it stands for t-Distributed Stochastic Neighbor Embedding [Jaju 2017]. t-SNE is a non-linear algorithm used for dimension reduction of high-dimensional data. It generally maps the multi-dimensional data into a space which is observable for humans.

Since the t-SNE algorithm is kind of a black box, it is not feasible to get insights on the real algorithm. Furthermore the algorithm doesn't provide the same output data on each successive run, even when the user uses the same data over and over again. Therefore the algorithm is better to be used for exploratory data analysis or as input for another classifier. The big advantage of the t-SNE algorithm is that it is capable of retaining the local and global structure of the data at the same time, whereas many other algorithms cannot do so.

The code snippet in Listing 8.1 shows how easy it is to use t-SNE with the help of the Rtsne function for plotting the similarity matrix map.

Figure 8.2 shows an example of such a similarity map created by the CSA application with the t-SNE algorithm. Furthermore, when the user decides to use the t-SNE algorithm for the similarity map layout, he gets the possibility to adapt some parameters, the number of maximum iterations as well as the perplexity. Figure 8.3 shows the UI of the adaptable t-SNE parameters.

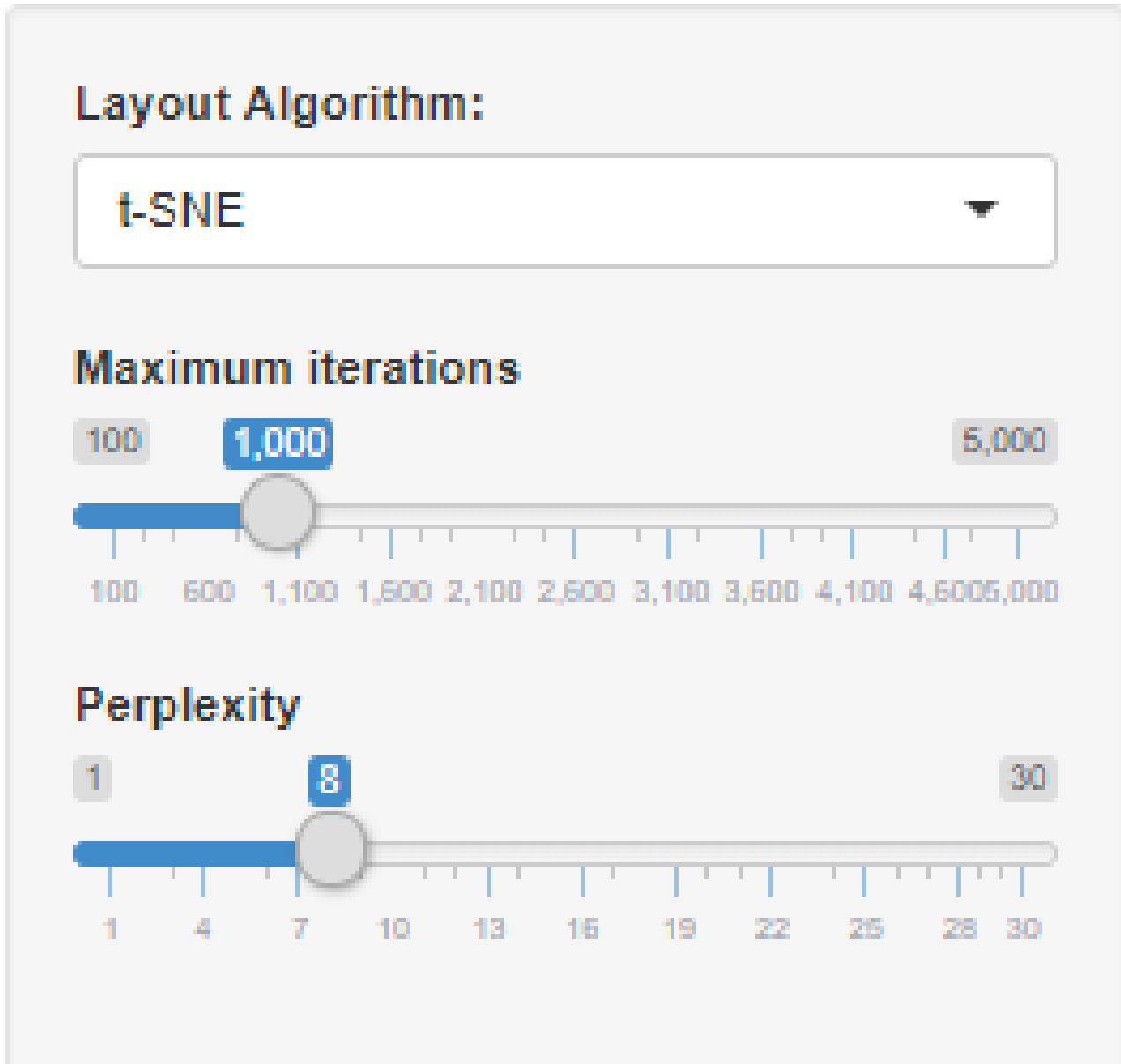


Figure 8.3: Parameters maximum iterations and perplexity which are used for the t-SNE Algorithm.

8.2 Similarity Map with MDS

As described in [Joseph B. Kruskal 1978], classical multidimensional scaling (MDS) is a method to project high-dimensional data into a lower-dimensional space in order to visualise the "hidden structure" of the data. The MDS algorithms use proximities among elements as input and provide a geometric representation of these elements as output, whereby the geometric representation reflects the proximities of the elements.

In the CSA application a two-dimensional representation of the proximities of the cards is created by utilising the `cmdscale()` function, which is included in the base R stats package. According to Team [2015b] the `cmdscale()` function applies classical multidimensional scaling (MDS), also known as principal coordinates analysis (PCoA). The function takes a distance matrix (that is a set of dissimilarities) and returns a set of points where the distances between the points are similar to the dissimilarities of the elements represented by the points. A detailed specification of the `cmdscale()` function can be found in [Team 2015b].

The code snippet in Listing 8.2 shows how the `cmdscale()` function is utilised in the CSA application for

```
1 plotSimilarityMap = function() {
2   #in case the user's chosen algorithm is MDS
3   if(input$smType == "MDS") {
4     #apply the MDS algorithm
5     fit <- cmdscale(dist1(), eig=TRUE, k=2)
6     #extract the coordinates of the points in the 2D-Space
7     x <- fit$points[,1]
8     y <- fit$points[,2]
9     #create a scatterplot of the points
10    sim_plt <- plot(x, y,
11                  #display no axes and no axes-labels
12                  xlab="", ylab="", axes = FALSE,
13                  #display a frame and a diagram-title
14                  frame = TRUE,
15                  main="Similarity Map (MDS)",
16                  #do not display the points
17                  type="n")
18    #plot the card-names at the position of the scatterplot points
19    text(x, y, labels = colnames(mat()), cex=0.8, font = 1, family = "sans")
20  }
21  return(sim_plt)
22 }
```

Listing 8.2: The code for plotting the similarity map with MDS algorithm in the CSA application utilises the `cmdscale()` function from the base R stats package.

plotting the similarity matrix.

Figure 8.4 shows an example of such a similarity map created by the CSA application with the MDS algorithm.

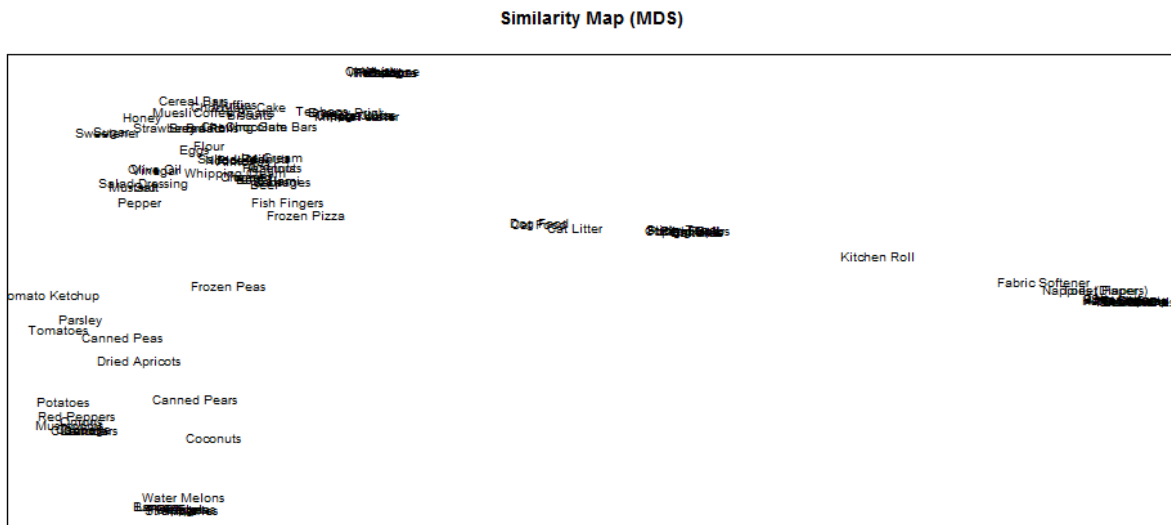


Figure 8.4: In the CSA application the similarity map graphic with MDS algorithm is created using the `cmdscale()` function, which comes with the default distribution of R.

8.3 Similarity Map with FDP

As described in Hu [2006] force-directed algorithms place the vertices of a graph according to a model of physical systems with forces acting between them, whereby the algorithm tries to minimise the energy of the system. There have been a lot of different force-directed algorithms developed. A widely used force-directed algorithm for drawing network graphs is the Fruchterman-Reingold algorithm. As stated by Fruchterman and Reingold in Thomas M. J. Fruchterman [1991], their design goal was to create an algorithm that keeps nearby vertices close together but not too close, whereby the algorithm should work well without the user having to sophisticatedly adjust parameters. For a comprehensive description of this algorithm refer to [Thomas M. J. Fruchterman 1991].

In the CSA application the `plot.igraph()` function from the R package `igraph` is used to create the similarity map graphic with force-directed placement (FDP) of the elements. A detailed specification of the `plot.igraph()` function is available on the R `igraph` manual pages [igraph Core Team 2015]. The `plot.igraph()` function provides several layout options, one of which is a force-directed placement using the Fruchterman-Reingold algorithm mentioned above. The code snippet in Listing 8.3 shows how the `plot.igraph()` function is utilised in the `csa` application for plotting the similarity map with force-directed placement.

Figure 8.5 shows an example of such a similarity map with force-directed placement created by the `CSA` application.

```
1
2 plotSimilarityMap = function() {
3   #in case the user's chosen algorithm is FDP
4   if (input$smType == "FDP") {
5     #calculate the normalised similarity matrix
6     simuMat <- (nrOfLines() - csdistance())/ nrOfLines()
7     rownames(simuMat) <- colnames(mat())
8     #create an undirected igraph graph from the normalised similarity matrix
9     g <- graph_from_adjacency_matrix(simuMat, mode = "undirected",
10                                     weighted = TRUE, diag = FALSE)
11     #Use the force-directed layout algorithm by Fruchterman and Reingold
12     lo <- layout_with_fr(g)
13     #plot the igraph
14     sim_plt <- plot.igraph(g, layout = lo, niter = 100000,
15                            #do not display vertices and edges
16                            vertex.shape = "none", edge.lty = "blank",
17                            #display card names (i.e. rownames of similarity matrix)
18                            vertex.label = rownames(simuMat),
19                            vertex.label.color = "black",
20                            vertex.label.cex=0.8,
21                            vertex.label.font = 1,
22                            vertex.label.family = "sans",
23                            #plot a frame and diagram title
24                            frame = TRUE, main = "Similarity Map (FDP)")
25   }
26   return(sim_plt)
27 }
```

Listing 8.3: The code for plotting the similarity map with FDP in the CSA application utilises the `plot.igraph()` function from the R `igraph` package.

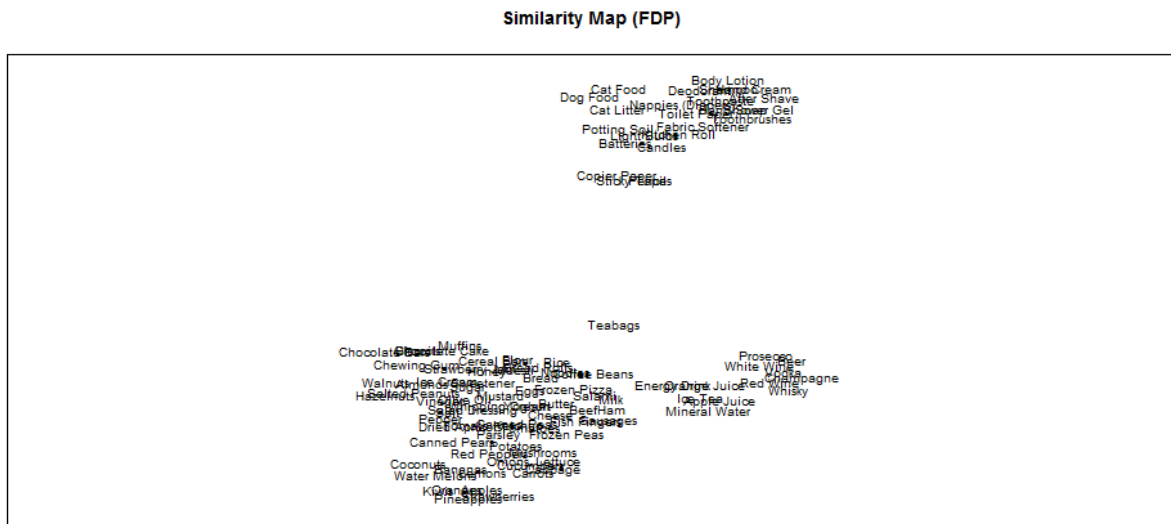


Figure 8.5: In the CSA application the similarity map graphic with force-directed placement (FDP) is created using the Fruchterman-Reingold layout option of the `plot.igraph` function from the `Rigraph` package.

Chapter 9

Dendrogram

The tab "Dendrogram" of the CSA application shows a dendrogram of the given data. A dendrogram is a tree diagram which is used to illustrate how the data is clustered depending on its clustering method. The user has the option to adapt the number of groups he wants to see as well as setting the wanted clustering method. Furthermore it is possible to download the dendrogram as png or as svg.

Figure 9.1 shows a screenshot of the "Dendrogram" tab with 12 groups and the "average" method chosen for clustering.

CSA

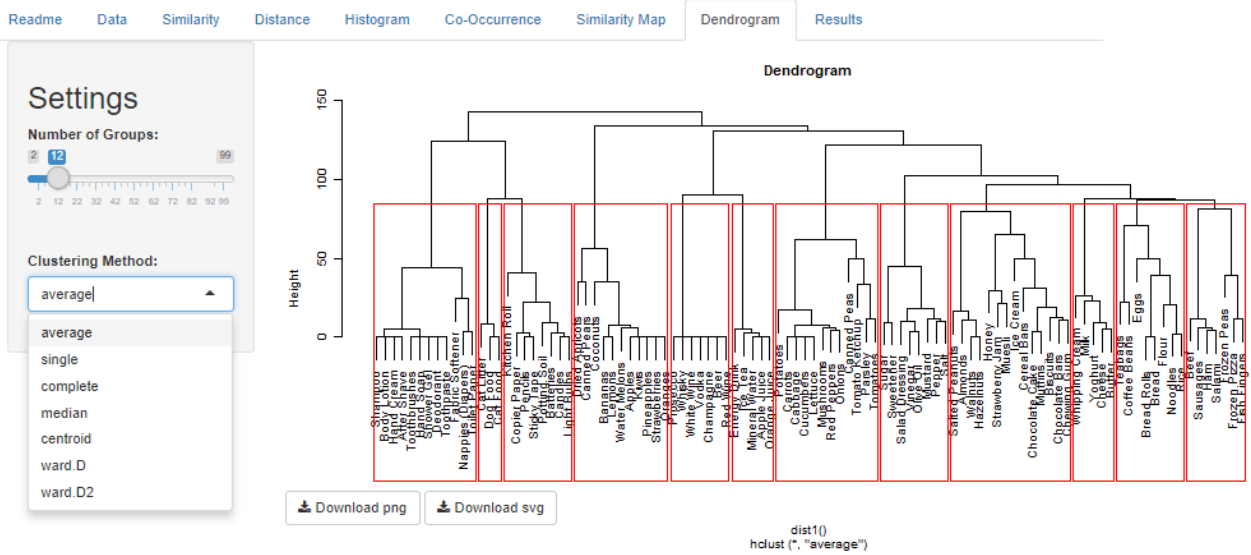


Figure 9.1: The tab "Dendrogram" of the CSA application shows the dendrogram with adaptable number of groups and different clustering method possibilities.

Chapter 10

Grouped Results

In this Tab the items are placed into groups, based on the number of groups chosen in the Dendrogram Tab. More groups mean more specialization, less groups result in a more general split. The number of groups has to be chosen carefully, with respect to the data set.

New functionality in this screen includes:

- Render multiple groups side by side
- Add border around group tables for better separation
- Calculation of the "best" group names

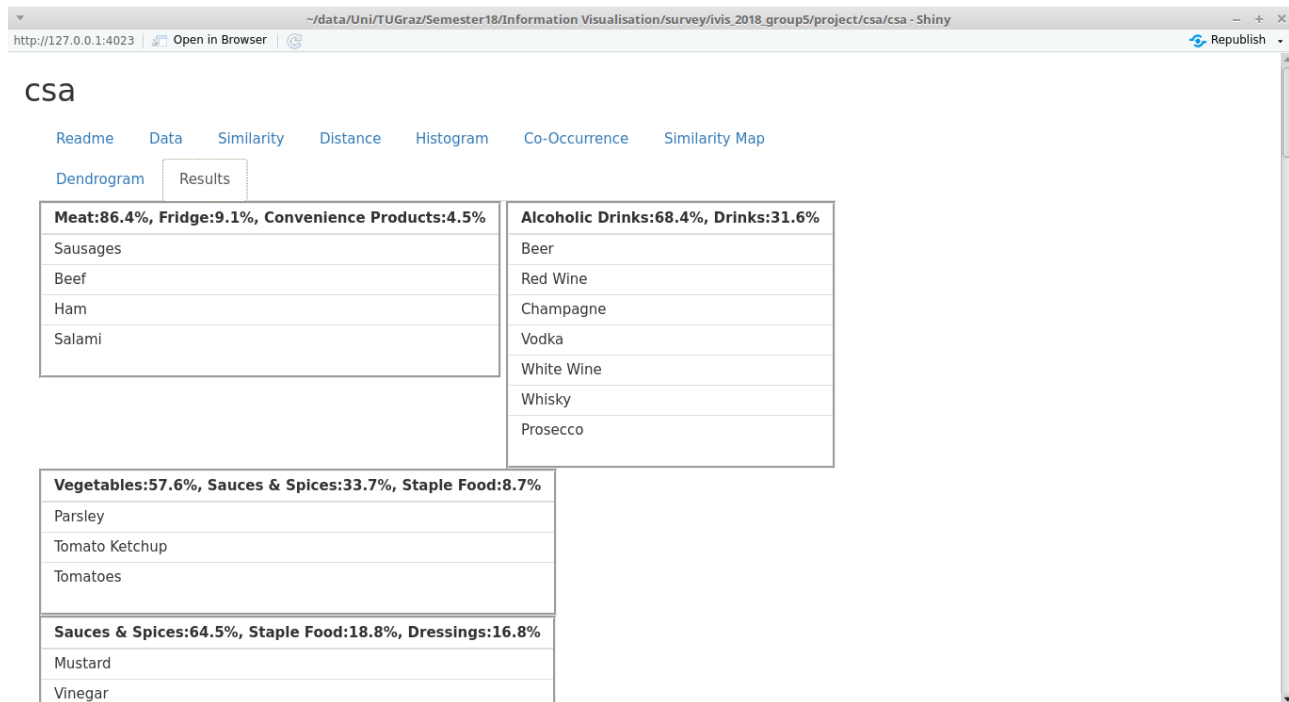


Figure 10.1: The tab "Results" of the CSA application shows the grouped results, with the 3 most probable group names.

```

1 countList = list() #our count dictionary, where the name of the entry is the unique key
2 for(index in 1:nrow(data)){
3   item <- paste(data[index, 1]) #get entry as String
4
5   votes <- fullData[, item]; #get column data of csv where title = our current item
6   for(v in votes){
7     vote <- paste(v)
8     #count how many people voted for which category for this item
9     if(vote %in% names(countList)){
10      countList[[ vote ]] = (countList[[ vote ]] + 1)
11    } else {
12      countList[[ vote ]] = 1
13    }
14  }
15 }
16 #get first 3 entries of sorted count list
17 numberOfCat <- 3
18 mostProbableNames <- head(sort(unlist(countList), decreasing = TRUE), numberOfCat);
19
20 #Calculate percent and format text, show x most probable group names as header with
    percent values
21 totalCount <- Reduce("+", mostProbableNames)

```

Listing 10.1: Calculation of the 3 most probable group names, based on the counts of groups for each item.

Chapter 11

Conclusion

Compared to the previous version of the CSA software there were several fixes and adaptations made as well as some totally new features were implemented. Starting from automatically reading and displaying the number of lines of the input file instead of forcing the user to provide it manually, over to having the option to display normalised data ("Similarity" and "Distance" tab), making the histogram more beautiful, adding another similarity map layout algorithm, providing the most likely group names as well as having several options to download the displayed data.

Nevertheless there is still room for improvement. Due to the fact that Shiny is very interactive and constantly processing, the performance of the application is not as high as one could imagine for such a relatively simple application. Since there are no huge coloured pictures which need to be rendered interactively, some adaptations regarding the speed of the software can be made. Another possibility would be to make the whole application self-explaining by providing information to each tab so that the user immediately knows the background of the algorithms/images which are being displayed.

Bibliography

- Hu, Yifan [2006]. “Efficient, High-Quality Force-Directed Graph Drawing”. *The Mathematica Journal* 10.1 [2006], pages 37–71. http://www.mathematica-journal.com/issue/v10i1/contents/graph_draw/graph_draw.pdf (cited on page 26).
- Igraph Core Team, The [2015]. *R igraph Manual Pages*. igraph.org. 2015. <http://igraph.org/r/doc/> (cited on page 26).
- Jaju, Saurabh [2017]. “Comprehensive Guide on t-SNE algorithm with implementation in R & Python” [2017]. <https://www.analyticsvidhya.com/blog/2017/01/t-sne-implementation-r-python/> (cited on page 22).
- Joseph B. Kruskal, Myron Wish [1978]. *Multidimensional Scaling*. 11th Edition. SAGE, 1978. ISBN 0803909403 (cited on page 24).
- Team, R Core [2015a]. *The R Graphics Package - Documentation for package 'graphics' version 3.6.0*. Seminar für Statistik, ETH Zürich, Switzerland. 2015. <https://stat.ethz.ch/R-manual/R-devel/library/graphics/html/hist.html> (cited on page 15).
- Team, R Core [2015b]. *The R Stats Package, Version 3.6.0*. Seminar für Statistik, ETH Zürich, Switzerland. 2015. <https://stat.ethz.ch/R-manual/R-devel/library/stats/html/cmdscale.html> (cited on page 24).
- Thomas M. J. Fruchterman, Edward M. Reingold [1991]. “Graph Drawing by Force-directed Placement”. *Software - Practice and Experience* 21.1 [11 1991], pages 1129–1164. doi:10.1.1.13.8444. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.8444&rep=rep1&type=pdf> (cited on page 26).