Module 4 Making sense of relational and textual data

Kimbal Marriott

Contents

Chapter 1

Making sense of relational and textual data: Overview

By Kimbal Marriott

Updated 19 March 2018

This is the fourth module in FIT5147 Data Exploration and Visualisation. In this module you will learn about how to explore and visualise textual data and data that is organised into networks or hierarchies.

Aims of this module

After completing this module you will:

- have seen standard visualisations for showing hierarchies, processes, sets and networks and understand when it is appropriate to use them;
- know statistical measures commonly used for network analysis and different kinds of networks;
- know standard analysis and visualisation techniques for individual text documents and for document corpuses;
- have first-hand experience with using R and the packages **igraph** for network analysis and visualisation and **tm** for text analysis;
- have first-hand experience using Voyant Tools for text analysis.

How to study for this module

In this module we draw on material in the public domain, including journal articles and some videos.

Chapter 2

Network analysis and visualisation

By Kimbal Marriott

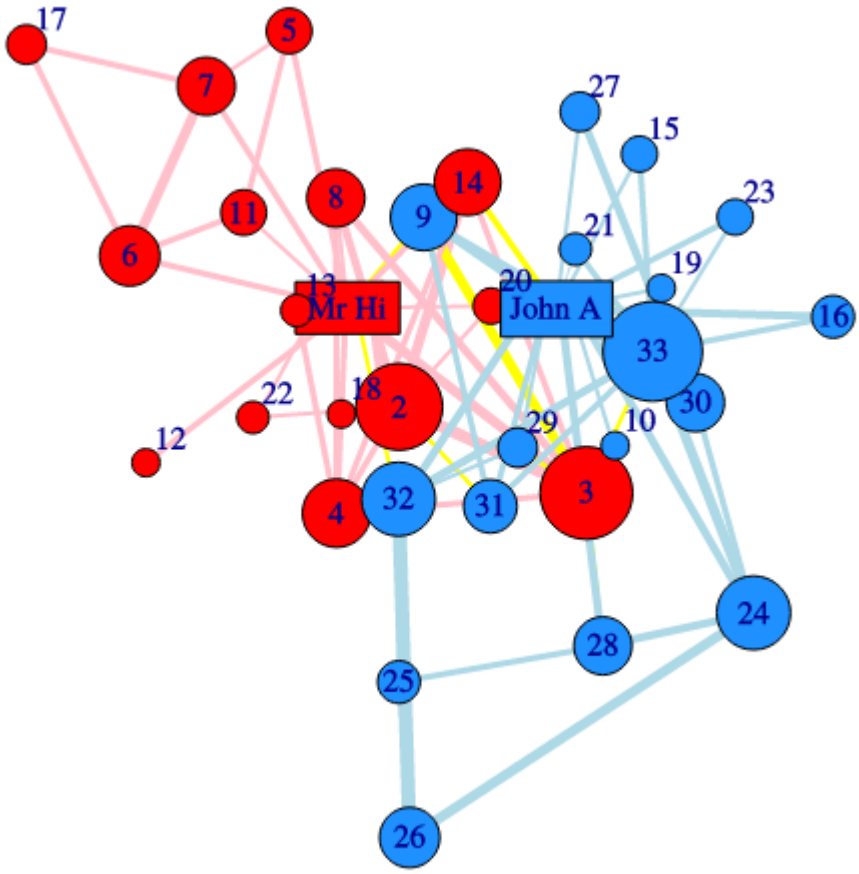
Updated 9 March 2017

Networks are well-suited to modelling abstract relationships between entities. *Social networks* (or *sociograms*) are one of the best known kinds of networks, these model communities and show the strength of connection between different people in the community. Networks are also common in software engineering where they are used to show the structure of software or its execution, and in the life sciences where they are used to model processes, for instance metabolic pathways.

Please take a look at the TED Talk by Nicholas Christakis on the The hidden influence of social networks (21 min) to see great examples of social network analysis and visualisation.

Our running example in this topic will be the "karate club network" of Zachary. This shows the amount of social interaction between members of a karate club during the time the club split into two clubs because of differences between one of the head teachers and the administrator.

Figure 2.1: The network is from: https://github.com/igraph/igraph/blob/master/doc/presentations/infovis07/plots.R



And here is the network laid out to show the two factions (Mr Hi, '1' and John A, '34') with connections

within factions shown in red or blue and inter-faction connections in yellow. The width of each edge is proportional to the strength of interaction and the size of the vertex is proportional to the total width of the edges entering it. (from: https://github.com/kolaczyk/sand)

Networks are becoming a much more common way of organising data because of the rise of graph databases such as Neo4j. While traditional relational databases support tabular data, graph databases support networks. Graph databases have the advantage that they scale up to larger data sets and are less rigid allowing data to be stored in more ad hoc relationships which can evolve over time.

2.1 Some terminology

Networks are formalised as multivariate graphs or networks. These have *nodes* (also called *vertices*) which correspond to entities such as people or businesses, *edges* (sometimes called arcs) which connect two nodes and which represent a relationship such as "is-an-employee" and *attributes* or *properties* of either nodes or edges such as "years-employed" or "age." The edges are said to be *weighted* if they have a strength or magnitude.

A network is *dynamic* if it changes over time.

The network is *directed* if the edges have a direction: a diagram showing the inheritance relationship between classes in C++ or Java code is an example of a directed network. The karate club is an example of an undirected network.

The *neighbours* of a node are the nodes that are connected to it by a single edge. A path between two nodes is a sequence of edges between the two of them. It is the *shortest path* if it has the least length. The length of the shortest path between two nodes is the (*graph-theoretic*) *distance* between them. In a social network this is called the *degree of separation*. The *diameter* of a network is the longest shortest path between any two nodes in the network. Two nodes are *connected* if there is a path between them, otherwise they are *unconnected*. A path is *directed* if the edges in it are directed and flow from the start to the end of the path.

One apparent limitation of a network is that it can only represent binary relations. A network in which edges can connect more than two edges is called a *hyper graph* and the edges are called *hyper edges*. However, it turns out that for most practical purposes hyper graphs are unnecessary as they can be modelled using a standard binary edge network by introducing a new node corresponding to each hyper edge and linking this node to each of the original nodes connected by the hyper edge using a binary edge.

2.2 Network visualisation

The most popular way of representing a network is using a *node-link diagram* like that showing the karate club social network. These place the nodes on the drawing canvas and use lines between them to represent edges. The edges are typically drawn as straight lines but can be curved edges, have multiple straight-line segments, or be drawn using only horizontal and vertical segments in the style of a electrical circuit.

Creating a good node-link diagram is not easy. Edges should not cross nodes and a good diagram should try and reduce edge crossings, have straight edges, have roughly uniform edge length, show symmetry and the graph structure. Unfortunately these criteria conflict so there is a trade-off between them. Furthermore, even optimising one of the criteria such as reducing edge crossings is an NP hard problem so automatic network layout is computationally a very hard problem.

Fortunately many different layout algorithms have been developed for node-link diagrams. The most common approach are the so-called *force directed approaches*. Many are loosely based on a physical model of the network in which the nodes are spheres that repel each other and the edges are springs with a natural length that connect the spheres. The layout is found by using optimisation to find a placement for the nodes that minimises the total energy in the physical model. Closely related are *multi-dimensional scaling (MDS)* based approaches. In this approach the distance between each pair of nodes is set to be the graph theoretic distance between them: thus if they are neighbours the distance is 1. If there are N nodes then this gives a distance

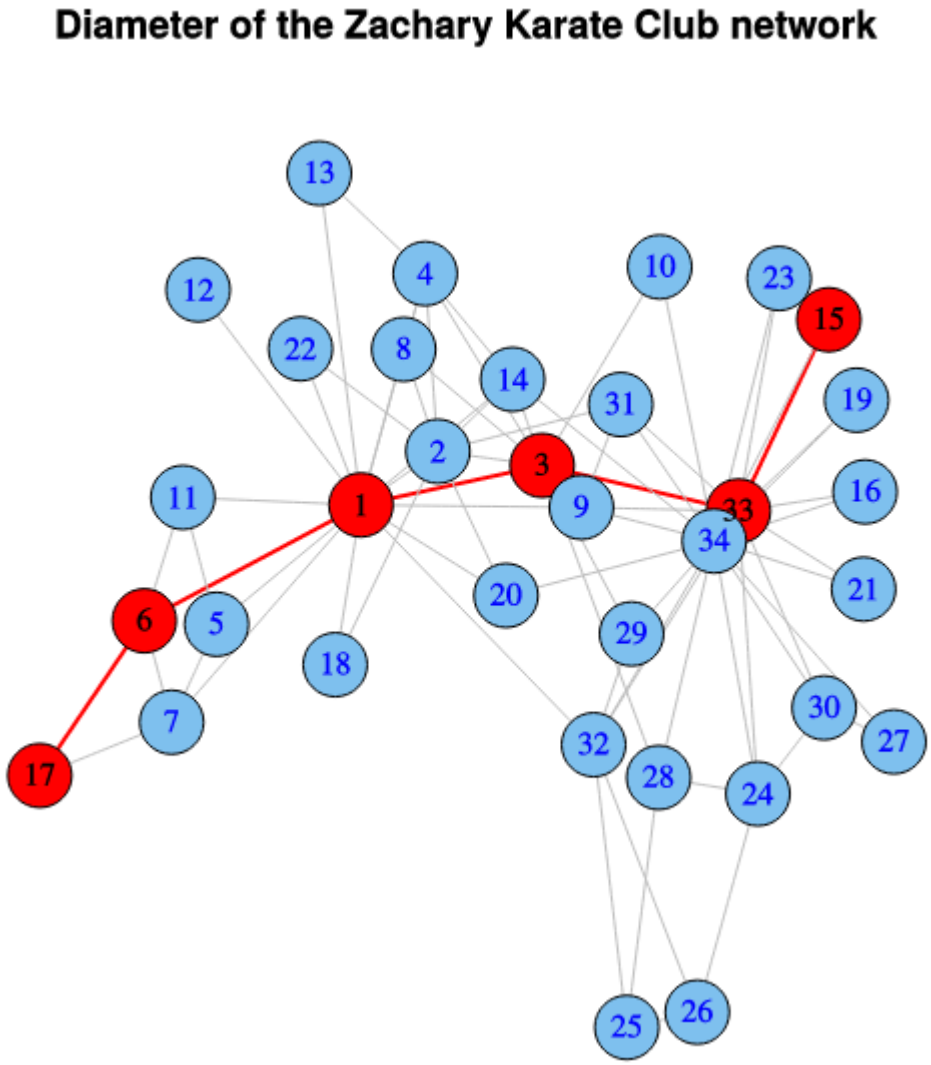
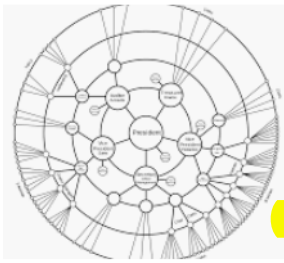


Figure 2.2: From: https://github.com/igraph/igraph/blob/master/doc/presentations/infovis07/plots.R

between all of the nodes in N dimensional space. To find the network drawing MDS techniques are used to project this space onto two dimension so as find the placement that best preserves this graph theoretic distance. The Kamada-Kawai algorithm is an MDS approach that uses a force directed algorithm to find the positions. It was used for the above layouts.



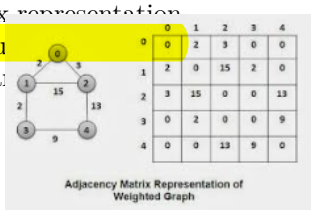
ected and MDS-based approaches place the nodes anywhere in the plane. Other drawing styles he placement opti ns. In a *chord diagram* the nodes are placed on the perimeter of a circle and d by internal curved edges while in an *arc diagram* the nodes are placed on a line but also connected l edges.

se of networks with directed edges it is common to use some kind of *layered diagram* in which s are aligned in bands and edges that (generally) flow from left-to-right or from top-to-bottom. A *adial network diagrams* in which the edges radiate outwards from a central node and the layers ic circles.

Representing networks using some kind of node-link diagram is by far the most popular way of showing the network. Node-link diagrams have the advantage that most people are familiar with them and for small networks they support many of the abstract tasks that are important for network data such as finding paths between nodes or understanding the structure. However for larger networks, particularly for networks that are dense in the sense that they have lots of edges, edge crossings and general edge clutter make it difficult to understand the network structure and to distinguish edges. While techniques like edge-bundling or grouping nodes with common edges in so-called power graphs reduces this clutter, after a certain network size and node density, node link diagrams are very difficult to understand.

An alternative visualisation which has the advantage of scaling to larger networks is the *adjacency matrix* representation. If there are N nodes than the matrix is N x N and the entry in cell (i, j) if there is an edge from node i to node j. Of course for undirected networks the matrix will be symmetric so only one of the diagonal halves needs to be shown.

For the adjacency matrix representation to be effective the rows and columns should be reordered so as to place connected nodes next to each other and to try and create rectangular "blobs." To preserve the reader's orientation the row and column ordering should be consistent. With this representation tightly connected clusters can be seen as "blobs" down the diagonal. The disadvantage of the adjacency matrix representation is that, at least for small graphs, it does not show the overall structure as clearly, it is difficu between nodes, and it is unfamiliar to most people. On the other hand it has the great adva clutter is completely removed.



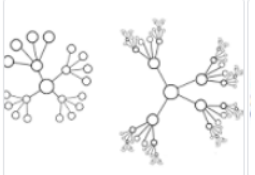
2.3 Network analytics

There are a number of common statistical measurements for characterising the structure of the network and the importance of nodes and edges.

One fundamental measure is the *degree* of a node. This is the number of edges incident to the node (or equivalently the number of neighbours). We can plot the distribution of the node degree. In the case of weighted graphs, node degree is generalised to *nodestrength* which is the sum of the weights of the incident edges.

The frequency of node degree can give insight into the underlying graph. *Scale-free networks* are a common kind of network in which the degree distributions follows a power law. In other words the number of nodes having degree k in the network is proportional to k where is typically between 2 and 3. The network of citations between scientific papers forms a scale free network, with some papers being massively cited.

Another interesting class of networks are the so called *small-world networks*. In these networks the average distance between two nodes grows logarithmically in the size of the network rather than linearly. Social networks are often claimed to be small-world networks as almost everyone has friend or a friend of friend in common. Many small-world networks are scale-free.



Centrality

A major focus in network analytics is to measure the importance or *centrality* of a node n. There are a variety of approaches:

- The simplest measure is the *degree* of n: the higher the degree the more important n is.
- *Closeness centrality* captures the idea that a node n is central if it is close to many other nodes: this measure varies inversely with the total of the node's graph theoretic distance to all other nodes in the graph.
- *Betweenness centrality* summarises the extent to which the node n lies between the other nodes: it is based on counting the number of shortest paths that pass through n.
- Status or rank centrality is based on the assumption that a node is more important if it has important neighbours. These are found using eigenvalues and eigenvectors.

If you are exploring node centrality it is common to use a node-link diagram in which nodes are placed radially with the nodes of most centrality at the center.

We can also consider the centrality of an edge e. Here the most obvious measure is *betweenness centrality*: this is the number of shortest paths that pass through e.

Network cohesion

Another major focus in network analytics is *network cohesion*. This basically investigates how tightly connected the graph is.

- One approach is to measure the number of highly connected sub-graphs. A *clique* is a subset of nodes for which there is an edge between each pair of nodes in the subset while a *bi-clique* is two subsets of nodes such that every node in each subset has an edge to the nodes in the other subset. One network analytic measures the number of different sized cliques and bi-cliques in the network. Closely related is the *k-core*: a subset of nodes in which each node has at least k edges to other nodes in the subset.
- *Density* captures the frequency of actual edges compared with the total number of edges. The density of a subgraph is $\frac{M}{N*(N-1)/2}$ where M is the number of edges and N the number of nodes.
- The *clustering coefficient* measures the frequency in which sub-graphs with 3 connected nodes are actually cliques, i.e. there is an edge between all 3 nodes.
- *Vertex* and *edge connectivity* measure how many nodes or edges need to be removed from a connected network before it is disconnected.

Assortative mixing

Assortative mixing looks at how the attributes of a node affect the kinds of node it is linked to. For instance, in a social network this might reveal that gender affects friendship and that men are more likely to be friends with men and women with women. If the nodes can be divided into K categories based on their attribute values then assortative mixing computes the assortativity coefficient for each pair of categories. This ranges between -1 and 1. It is 0 if the category does not affect the frequency of edges and 1 if edges only occur between these two categories.

Graph clustering and partitioning

The final focus in network analytics is how to cluster the nodes in a graph. This is also called *community detection*. The aim is to partition the graph into subgraphs so that each subgraph is highly connected while the subgraphs are loosely connected.

- *Hierarchical clustering* techniques can be used for this. One approach is to identify small highly connected subgraphs and then repeatedly merge the most highly connected subgraphs. This produces a hierarchy of nested partitions for which a dendrogram can show the hierarchy and the "connectedness" between sub-components.
- *Spectral partitioning* uses eigenvalues to recursively split the graph into smaller components.

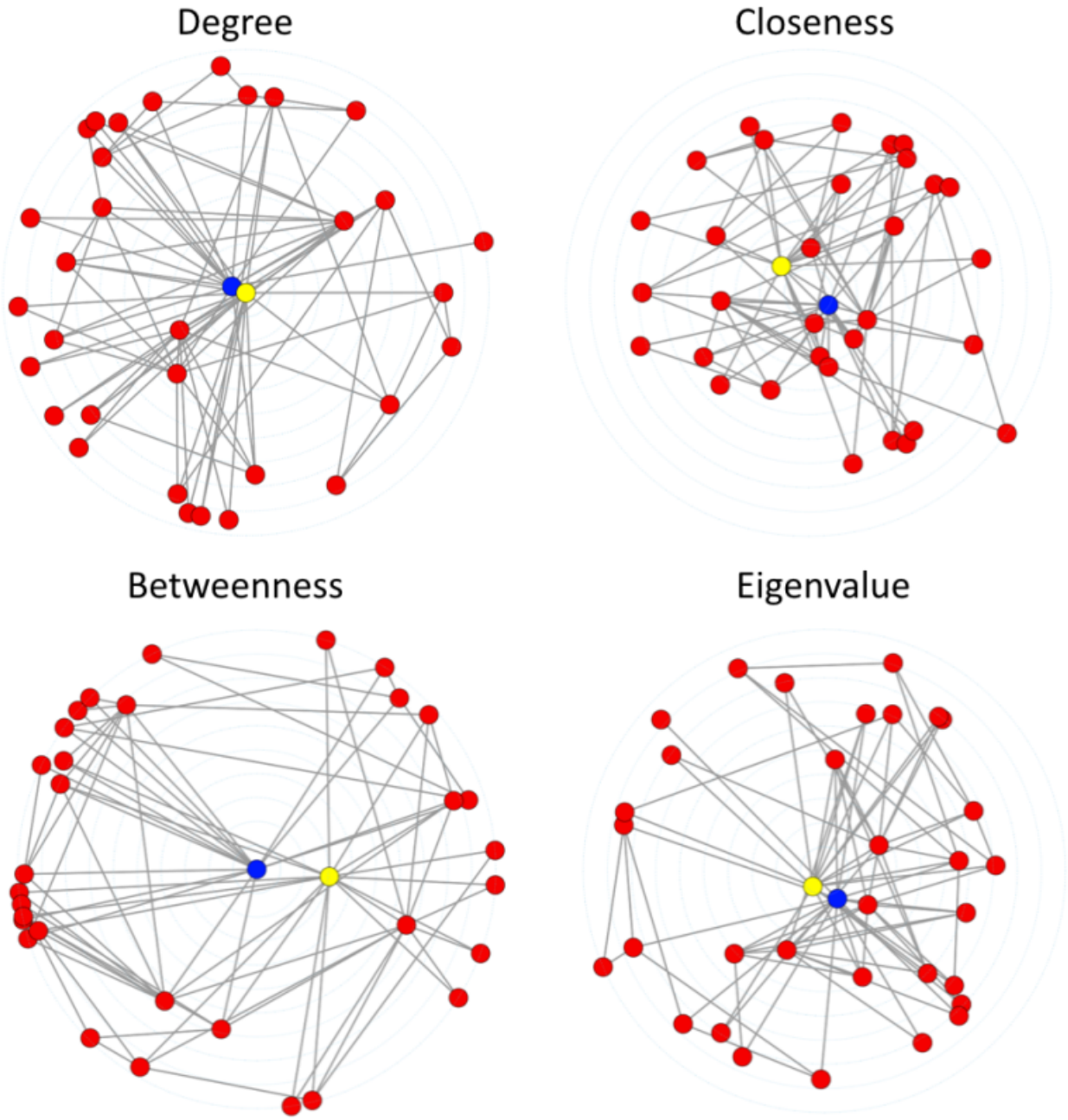
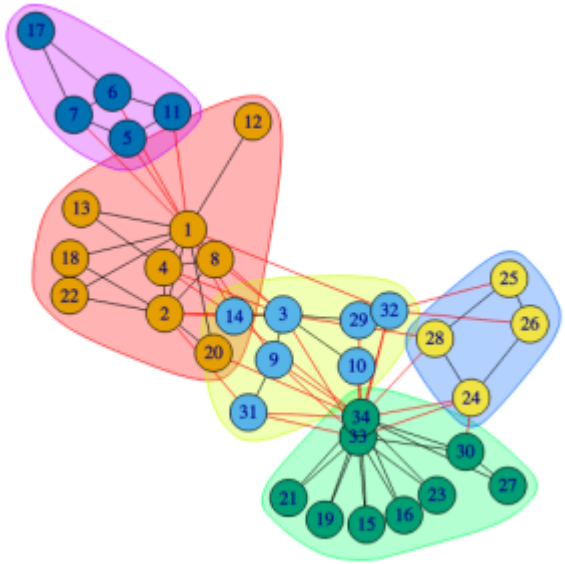


Figure 2.3: Different centrality metrics for the Karate Club Network (Based on Figure 4.4 of *Statistical Analysis of Network Data with R.*)

Clustering plays an important role when exploring networks but the results should always be visualised to see if they make sense and also if analysis of the attributes etc can help to explain the partitioning.



The "karate club network" with clustering. From: http://igraph.org/r/doc/communities.html

2.4 Trees and hierarchies

Trees are a common kind of network in which there are no cycles, i.e. there is at most one path between each pair of nodes. If the edges are directed then the tree is called a *hierarchy*. In a true hierarchy for each node n there is at most one node m with an edge from m to $n.m$ is called the parent of n. An organisation structure is often a hierarchy as is the subclass relationship when multiple inheritance is not allowed.

There are a number of quite different ways of drawing hierarchies. These rely on different visual or spatial relationships for showing the parent relationship. One of the most common ways is to use an indented list: file directories are often shown like this.

One example of a layered tree that we have met throughout this unit is the dendrogram. They are widely used to show similarity between hierarchical clusters with the vertical distance between parent and child proportional to their difference. Cladograms are very similar to dendrograms but show the evolutionary history of related species.

However using lines to show the parent relationship is not the only possible visual metaphor. Icicle plots and sunburst plots use adjacent overlap to indicate the parent relationship while tree maps use spatial containment. Tree maps are widely used to show hierarchically organised quantitative data, with the size of the rectangle being proportional to the data value. While they are visually appealing we have found that using an indented list with an associated bar chart also works quite well and is easier to navigate around.

Containment is commonly combined with node-link representations when showing networks whose nodes have a hierarchical structure. This structure might be given or inferred by network clustering.

2.5 Flow diagrams

Networks are also used to show flow and processes. Flow diagrams show the steps or events in a process. An example of flow diagrams are flow charts. These were once widely used to show flow of control in algorithms and business processes.

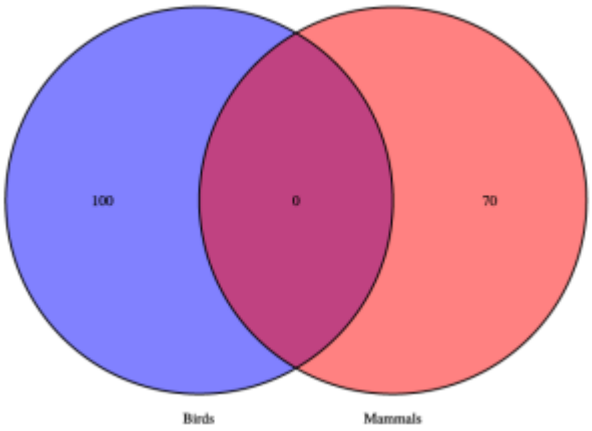


Figure 2.4: Venn diagram, empty intersection, not proportional area (both created using the R library VennDiagram)

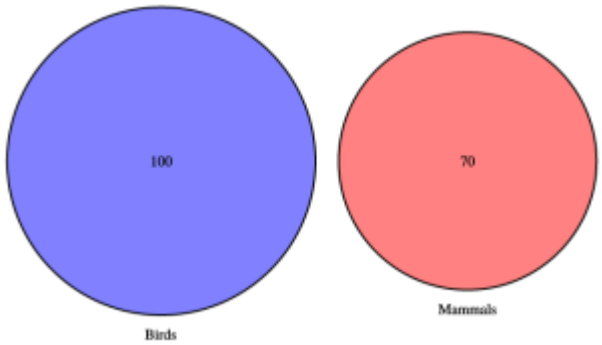
Another kind of flow diagram is the Sankey diagram which shows the amount of flow between different processes using variable width arrows or lines where the width is proportional to the amount of flow. They are similar to flow maps except that flow is between abstract entities rather than geographic locations.

One problem with Sankey diagrams and other flow diagrams is that, like all node link diagrams, for large dense networks the diagram becomes cluttered and confusing because of edge crossings and overlap. In such cases matrix representations may be a better approach. Recent research in this area has investigated matrix representations like MatrixWave for showing abstract flow and MapTrix for showing flow between geographic locations.

2.6 Set visualisation

An important class of abstract relationships are based on set containment. Specialised kinds of diagrams have been developed to show the relationship between sets and their union and intersection. The best known are Venn and Euler diagrams. These use spatial containment to represent set containment.

Venn diagrams differ from Euler diagrams in that the diagram must contain all possible set intersections, with shading showing which intersections are empty. Euler diagrams do not show empty set intersections. In proportional area Euler diagrams the size of the regions is proportional to the data quantity, such as number of members or probability. These are used in medicine.

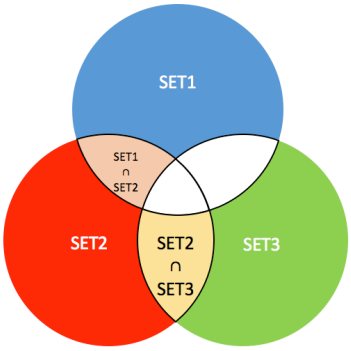


Proportional area Euler diagram with no intersection (100:70)

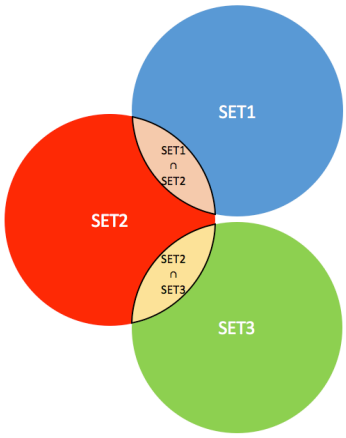
Like node-link representations for networks Venn and Euler diagrams are intuitively easy to understand and allow small data sets to be readily understood. However once the number of sets increases they are difficult

to draw and become extremely difficult to understand. Again a more abstract matrix representation provides a scalable alternative but one that is unfamiliar to most people. This matrix representation is the *linear diagram*. There is a row for each set and a column for each non-empty intersection of sets. The column width can be set to the size of this intersection.

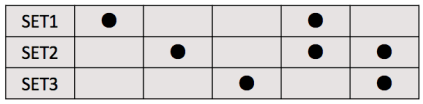
Here are the same 3 sets drawn using these three different representations.



Venn Diagram



Euler Diagram



Matrix presentation for set visualisation

2.7 Summary

In this topic we have investigated the main kinds of visualisations for showing abstract data such as networks, hierarchies, control and process flow as well as set relationships. Most of these visualisations rely on using some kind of visual or spatial metaphor for the abstract relationship. For instance node-link diagrams using a line to show two entities are connected while Euler diagrams use spatial containment to show set containment.

The use of such visual and spatial metaphors means that the visual representations are intuitive and easy to understand. However these metaphors may not scale well to larger data sets. For larger data sets matrix based representations like the adjacency matrix for networks are often better. Their main disadvantage is that they are unfamiliar to most readers.

We have also looked at clustering and common statistical measures such as centrality and cohesion for understanding networks. These are an important part of the data scientist's toolbag. Originally developed for social networks they are useful in understanding the rich and complex interrelationships between more abstract data.

FURTHER READING

Take a look at the TED Talk by Manuel Lima. A visual history of human knowledge on the history of tree and network visualisation which contains some really beautiful visualisations.

This topic is based upon

Eric D. Kolaczyk and Gábor Csárdi. *Statistical Analysis of Network Data with R*. Springer 2014

Kerren, Andreas, Helen C. Purchase, and Matthew O. Ward (Eds). *Multivariate Network Visualization*. Springer International Publishing, 2014.

Ward, Matthew O., Georges Grinstein, and Daniel Keim. Chapter 9 of *Interactive data visualization: foundations, techniques, and applications (2nd Ed)*. CRC Press, 2015.

Munzner, Tamara. Chapter 9 of *Visualization Analysis and Design*. CRC Press, 2014.

Chapter 3

Activity: Relational data analysis and visualisation with R

By Laurens, Yalong Yang

Updated 17 March 2018

Relational/network data & visualizations

We can represent networks or relationships (think of a family tree, or social groups) as graphs. Starting with Part A, basic graphs, then Part B, adding direction (e.g. 'friend' may not have a known direction but 'child' does) then looking at some analytics (Part C), i.e. who are the people in the centre of a network (think the office manager in an organisation), how strong are connections, who is or is not connected to whom.

We're going to introduce and look at the following aspects of networks:

- nodes
- edges
- direction
- weights
- degree
- diameter
- sub graphs

Networks or graphs or network graphs have nodes (also called vertices), usually things like people and edges (sometimes called arcs) which connect two nodes and which represent a relationship such as "friend" or "owes-money". The edges are said to be weighted if they have a value. e.g. A owes B $10.

A network graph is called directed if the edges have a direction e.g. I owe you money, you owe him money, he owes me money, (let's just call the whole thing off).

Which you could represent as: me->you, you->him, him->me

A. Basic graphs

We use the igraph library http://igraph.org/r/

Start by specifying a graph using numbers to represent nodes such that a pair of numbers means a connection e.g:

- 1-2, 1-3, 2-3, 3-4

Which would give us 4 nodes where 1 has and edge with (is connected to) 2 and 3, 2 to 3, and 3 to 4.

The first thing you want to do is to store this data in R.

Here are two different way of presenting this data in R (you will have the same result):

```
library(igraph)
g <- graph.formula(1-2, 1-3, 2-3, 2-4) # the connections, 1 to 2, 1 to 3 etc.
# or g <- graph(c(1,2,1,3,2,3,2,4), directed = FALSE)
```

And display

```
plot(g)
```

Let's investigate the important properties of a graph.

Vertices first, should be 4

```
V(g)
```

Edges then, should be also 4

```
E(g)
```

What are other important properties?

Diameter, of course.

If you do not know what it is, have a look at: http://stackoverflow.com/questions/3174569/what-is-meant-by-diameter-of-a-network

Do not confuse it with circle diameter, although they have some similarities.

```
diameter(g)
```

Node-link is not the only way to present a graph.

Another common way is the **adjacent matrix**.

```
get.adjacency(g) # a matrix view
```

Nodes deserve to have names.

```
V(g)$name <- c("Adam", "Bob", "Con", "Doug")
# and replot using $name
plot(g, vertex.label = V(g)$name) # add labels
get.adjacency(g) # have a look at the new matrix
```

B. Directed graphs

In some use cases, **directions** are very important.

So, how to present the directed graph in R?

Direction is usually represented by an arrow in real life, here, let's pretend it means 'owes money to'

The syntax is a little strange in R, '1-+2' means 'edge from 1 to 2'

or you can use directed = TRUE then the order specifies direction (so 1-2 IS NOT THE SAME as 2-1)

```
dg <- graph.formula(1-+2, 1-+3, 2-+3, 2-+4) # so '1' owes '2'and '3' etc.
plot(dg)
```

Can you have both 1-2 and 2-1?

Who does '4' owe money to?

Again, we need names to make it more readable (interesting), alphabetically such that 'A' is 1, 'B' is 2 etc.

```
V(dg)$name <- c("Adam", "Bob", "Con", "Doug") # names!
plot(dg, vertex.label = V(dg)$name)
```

We now have the connections, but every connection seems the same.

What's the strength of the connection? **i.e. How much money does Adam owe Bob?**

```
is.weighted(dg)
```

There are no 'weights' or debts… yet

Add (random) values to the connections (the edges):

```
wdg <- dg # copy, wdg is going to be a weighted directed graph
E(wdg)$weights <- runif(ecount(wdg)) * 1000 # random debts, up to $1000
plot(wdg, vertex.label = V(wdg)$name, edge.width=E(wdg)$weights)
```

Oops, what happened!? What are these weights anyway?

```
E(wdg)$weights # as specified, random values from 0 to 1000
```

What happened is that we plotted edges, as lines, with widths up to 1000 (pixels), which is a mess, so be careful!

Now we scale them down.

```
# so scale, but we might lose our arrows...
plot(wdg, vertex.label = V(wdg)$name, edge.width=E(wdg)$weights / 100)
```

Do you like the way R put your nodes? Let's play with the layout.

There are lots of layouts in R.

Let's try star first.

```
plot(wdg, vertex.label = V(wdg)$name, edge.width=E(wdg)$weights / 100, layout = layout.star)
```

Actually, in RStudio, when you input the parameter: layout = layout. there will be a drop down box with different layout options.

Our debts graph is a bit basic (small) for layout testing, try a bigger one.

Note: some layouts have special requirement for the graph data, e.g. `bipartite`, look into them if you are interested!

```
# http://www.r-bloggers.com/going-viral-with-rs-igraph-package/

G <- graph( c(1,2,1,3,1,4,3,4,3,5,5,6,6,7,7,8,8,9,3,8,5,8), directed = F )

# Assign attributes to the graph
G$name <- "Change my layout, I dare you"

# Assign attributes to the graph's vertices
V(G)$name  <- toupper(letters[1:9])
V(G)$color <- sample(rainbow(9),9,replace=FALSE)

# Assign attributes to the edges
```

```
E(G)$weight <- runif(length(E(G)),.5,4)
# Plot the graph
plot(G, layout = layout.auto,
     main = G$name,
     vertex.label = V(G)$name,
     vertex.size = 25,
     vertex.color= V(G)$color,
     vertex.frame.color= "white",
     vertex.label.color = "white",
     vertex.label.family = "sans",
     edge.width=E(G)$weight,
     edge.color="black")
```

Your turn, change the layout

There are also a selection of built in layouts or types of graph e.g.

```
par(mfrow=c(2,2)) # 2 x 2 display

g <- graph.full(n=5, directed = FALSE, loops = FALSE)
plot(g)

g <- graph.star(n=5, mode="out")
plot(g)

g <- graph.star(n=5, mode="in")
plot(g)

g = graph.ring(n=5)
plot(g)
```

If you get an error like:

Error in .Call.graphics(C_palette, value) : invalid graphics state

Please click the "cross" button at above the plot area (beside the broomstick button) . If you get "plot region too large" in the plot area, please enlarge the plot area.

And we can add some colour:

```
par(mfrow=c(1,1))
g <- graph.full(5)
E(g)$weight <- runif(ecount(g)) # random weights, run again for different result
E(g)$width <- 1
E(g)$color <- "red"
E(g)[ weight < 0.5 ]$width <- 2
E(g)[ weight < 0.5 ]$color <- "blue"
plot(g, layout=layout.circle, edge.width=E(g)$width, edge.color= E(g)$color)
E(g)$weight
```

Sub graphs (select part of a graph):

```
g1 <- make_star(5)
g2 <- induced_subgraph(g1, 1:2) # select vertices
g3 <- subgraph.edges(g1, 1:3, TRUE) # select edges

par(mfrow=c(1,3))
```

```
plot(g1)
plot(g2)
plot(g3)
```

C. Analysis of Network Data

The Karate club

Explore the famous Karate club network e.g. who are the major players (or actors), the groups.

Let's look at the data first.

```
library(igraphdata)
data(karate) # load the built-in graph data
?karate
V(karate)
E(karate)
get.adjacency(karate)
```

Start by plotting the club in glorious colour with edges & nodes scaled to represent importance or weight.

1. Add label and define shape of vertices.

```
# Reproducible layout
set.seed(42)

# Now decorate, starting with labels and shapes
V(karate)$label <- sub("Actor ", "", V(karate)$name)
V(karate)$shape <- "circle"
```

2. Distinguish the faction.

```
# Differentiate two factions by color.
V(karate)[Faction == 1]$color <- "red"
V(karate)[Faction == 2]$color <- "dodgerblue"
```

3. Make the most popular one more obvious (make the circle proportional) **Investigate what is graph strength.**

```
# Vertex area proportional to vertex strength
# (i.e., total weight of incident edges).
V(karate)$size <- 4*sqrt(graph.strength(karate))
V(karate)$size2 <- V(karate)$size * .5
```

4. Make the connections proportional

```
# Weight edges by number of common activities
E(karate)$width <- E(karate)$weight
```

5. We definitely want to find out are there communications within/between faction?

Color the edges!

```
# Color edges by within/between faction.
F1 <- V(karate)[Faction==1]
F2 <- V(karate)[Faction==2]
E(karate)[ F1 %--% F1 ]$color <- "pink" # F1 to F1 i.e. internal
```

```
E(karate)[ F2 %--% F2 ]$color <- "lightblue"
E(karate)[ F1 %--% F2 ]$color <- "yellow"
```

6. If the vertex is too small, you cannot see the labels.

```
# Offset vertex labels for smaller points (default = 0). # or make all the circles bigger...
V(karate)$label.dist <- ifelse(V(karate)$size >= 10, 0, 0.75)
```

7. Make the layout and plot

```
l <- layout.kamada.kawai(karate)
plot(karate, layout=l)
```

If you do not like it, **try other layouts!**

Based on: https://github.com/kolaczyk/sand/blob/master/sand/inst/code/chapter3.R

Are there many within/between communications?

Centrality

A major focus in network analytics is to measure the importance or centrality of nodes:

- Degree – the simplest measure, the higher the degree the more important a node is (more edges = more connections).
- Closeness centrality – captures the idea that a node is central if it is close to many other nodes: this measure varies inversely with the total of the node's graph theoretic distance to all other nodes in the graph.
- Betweenness centrality – summarises the extent to which the node lies between the other nodes: it is based on counting the number of shortest paths that pass through n.
- Eigenvalues – Status or rank centrality is based on the assumption that a node is more important if it has important neighbours. These are found using eigenvalues and eigenvectors.

It is common to use a node-link diagram in which nodes are placed radially with the nodes of most centrality at the center.

Let's prepare the data first.

```
library(sand) # Statistical Analysis of Network Data with R
library(sna) # Tools for Social Network Analysis
library(network)

A = get.adjacency(karate, sparse=FALSE)

g = network::as.network.matrix(A) # make a matrix
```

Let's plot it.

These take a while to render… patience:

```
par(mfrow=c(1,1))
sna::gplot.target(
  g, degree(g), main="Degree",
  circ.lab = FALSE, # change to TRUE to see legend on concentric blue circles
  circ.col="skyblue", usearrows = FALSE,
  vertex.col=c("blue", rep("red", 32), "yellow"),
  edge.col="darkgray"
)
```

What's the blue circle, the yellow one?

Let's try other measurements.

```
par(mfrow=c(2,2))
par(mar=c(1.5, 1, 1, 0.5))

sna::gplot.target(
  g, degree(g), main="Degree",
  circ.lab = FALSE, # change to TRUE to see legend on concentric blue circles
  circ.col="skyblue", usearrows = FALSE,
  vertex.col=c("blue", rep("red", 32), "yellow"),
  edge.col="darkgray"
)

sna::gplot.target(g, closeness(g), main="Closeness",
                  circ.lab = FALSE, circ.col="skyblue",
                  usearrows = FALSE,
                  vertex.col=c("blue", rep("red", 32), "yellow"),
                  edge.col="darkgray")

sna::gplot.target(g, betweenness(g), main="Betweenness",
                  circ.lab = FALSE, circ.col="skyblue",
                  usearrows = FALSE,
                  vertex.col=c("blue", rep("red", 32), "yellow"),
                  edge.col="darkgray")

sna::gplot.target(g, evcent(g), main="Eigenvalue",
                  circ.lab = FALSE, circ.col="skyblue",
                  usearrows = FALSE,
                  vertex.col=c("blue", rep("red", 32), "yellow"),
                  edge.col="darkgray")
```

We can't see who's who so turn labels on with:

`displaylabels = TRUE`

https://www.uni-due.de/hummell/man/sna/.sna.gplot.pdf

```
par(mfrow=c(2,2))

sna::gplot.target(
  g, degree(g), main="Degree",
  circ.lab = FALSE, # change to TRUE to see legend on concentric blue circles
  circ.col="skyblue", usearrows = FALSE,
  vertex.col=c("blue", rep("red", 32), "yellow"),
  edge.col="darkgray", displaylabels=TRUE
)

sna::gplot.target(g, closeness(g), main="Closeness",
                  circ.lab = FALSE, circ.col="skyblue",
                  usearrows = FALSE,
                  vertex.col=c("blue", rep("red", 32), "yellow"),
                  edge.col="darkgray", displaylabels=TRUE)

sna::gplot.target(g, betweenness(g), main="Betweenness",
                  circ.lab = FALSE, circ.col="skyblue",
                  usearrows = FALSE,
```

```
                  vertex.col=c("blue", rep("red", 32), "yellow"),
                  edge.col="darkgray", displaylabels=TRUE)

sna::gplot.target(g, evcent(g), main="Eigenvalue",
                  circ.lab = FALSE, circ.col="skyblue",
                  usearrows = FALSE,
                  vertex.col=c("blue", rep("red", 32), "yellow"),
                  edge.col="darkgray", displaylabels=TRUE)
```

Edge centrality

Most network analysis focuses on the nodes/vertices but edges are important too.

Let's calculate the betweenness for edges.

```
eb = edge.betweenness(karate) # metric on the edges
```

What is betweenness?

Find out which edges are the top 3.

```
E(karate)[order (eb, decreasing = T)[1:3]]
# One by one
E(karate)[order (eb, decreasing = T)[1]] #
E(karate)[order (eb, decreasing = T)[2]] #
E(karate)[order (eb, decreasing = T)[3]] #
```

Have a look at the adjacency and look for '20 & '32'

```
par(mfrow=c(1,1))
G <- graph.adjacency(A)
plot(G) # actually it is hard to see
plot(G, vertex.size=5, layout = layout.davidson.harel) # still hard to see

plot(G, vertex.size = 5, vertex.label.dist = 0.5, edge.arrow.size = 0, layout = layout.kamada.kawai) #
# now look for '20' & '32'
```

Let's cluster the vertices into different groups by using the **edge measurements (betweenness and short random walks).**

```
par(mfrow=c(2,1)) # make two plots up and down
par(mar=c(0.5, 1, 1, 1))


kk.layout <- layout.kamada.kawai(G) # make a consistent layout for the graphs
# make the left plot
com <- edge.betweenness.community(G)
V(G)$color <- com$membership+1
plot(G, vertex.label.dist=.75,layout=kk.layout,
     main="Edge-Betweenness Communities",
     edge.arrow.size=0.25)

# make the right plot
com <- walktrap.community(G)
V(G)$color <- com$membership+1
plot(G, vertex.label.dist = .75, layout=kk.layout,
```

```
     main="Walktrap Communities",
     edge.arrow.size=0.25)
```

What is ‘walktrap’?

Chapter 4

Exploring text and document collections

By Kimbal Marriott

Updated 28 Feb 2018

Text analysis and mining is an increasingly important part of the data scientists job. The good news is that there is a lot of easily accessible data. Social media is one of the fastest growing data sources in the world. The first tweet was sent in March 2006 and by late 2015 about 500 million tweets a day are being sent. As well as twitter feeds, blogs, email, wikis, on-line news articles, web pages are ever growing sources of on-line text. The other big reason for the explosion in textual data is that libraries, governments and organisations are digitising their archives and placing the content on-line. There is now a mind boggling amount of really interesting textual data available on-line.

Please look at Jer Thorp's TedEd talk on Visualizing the world's Twitter data (6 min) to see some of the potential.

Now the bad news. Virtually all text is designed to be read by a human, not understood by a computer. Text is complex and only partially structured. This makes data analysis difficult and typically requires significant data wrangling before the analysis can start.

Because of the complexity of text wrangling and text mining we cannot hope to fully cover this subject here. We provide only a brief introduction to the most commonly used methods and techniques. However we do give references to more advanced treatments.

4.1 Preprocessing

When you read text in there are a number of basic preprocessing steps that you will probably need to do

- remove punctuation
- remove numbers
- convert to lower (or upper) case
- expand contractions like "it's"
- remove "stop words." These are common words that have little meaning and include "a", "the", "and" and so on.
- recognise compound words like "New Zealand" or "data science" and mark these to be treated as a single word
- stem words by removing common word endings such as "s" or "ed" or "ing."

More advanced preprocessing will perform spelling correction and slang replacement.

Figure 4.1: Word cloud of this chapter, NO text processing (so 'the' wins).

4.2 Exploring single documents

The most basic analysis is to count the frequency of terms in the document by which we mean those words left after stemming and removing stop words and contractions.

All the usual visualisations can be used to show term frequency. One common visualisation that is unique to text is to generate a *word* or *tag cloud*. These show the term itself with a size proportional to its frequency and are an oddly compelling way of summarising a document.

Another visualisation used to explore words in a single document is the word tree. The visualisation is centered on a target word chosen by the viewer. On the right side of the target words are arranged in a tree rooted on the target. The tree shows the most common sequences of words that follow the target word in the document. The first level shows the words that immediately follow the target with size proportional to how often they follow it. The next level shows the words that commonly follow these subsequences of two words and so on. A tree on the left can similarly be used to show the common word subsequences that precede the target.

Natural language Processing (NLP) techniques can be used to convert unstructured text into more structured data suitable for more advanced analytics.

Sentiment analysis aims to extract the attitude of the speaker or writer to the topic they are writing about. Generally this is classified as being positive, negative or neutral. Sentiment analysis is widely used by companies to track how they and their products are being written about in on-line reviews and social media.

Named entity recognition (NER) identifies the words in the text that refer to proper names such as places, organisations or people and which category the name is in.

Automatic summarisation can be used to get the gist of an article across in a few sentences or paragraphs. Extraction based methods do this by identifying the most important words, phrases and sentences in the document and presenting these in the summary. Abstraction based summarisation methods rely on more advanced NLP to perform a semantic analysis of the document and then summarise this abstract understanding. Search engines like Google use automatic summarisation when presenting search results.

Figure 4.2: Word cloud of this chapter, after English stopwords were removed (note 'word' and 'words' etc.) Both created using the wordcloud and tm libraries in R.

4.3 Exploring a document corpus

Often there is a collection of documents to understand. Such corpuses can be huge. One common task is to try and understand which documents in the corpus are *similar*. An extreme form of document similarity is when the documents contain near identical chunks of text. Fast algorithms for detecting shared sentences and paragraphs have been developed for *plagiarism detection*.

Apart from the special case of plagiarism detection the most widely used techniques for measuring document similarity are based on computing a vector of weighted word frequencies for each document. The same set of words is used across the document corpus. The distance between the document vectors gives the similarity between the two documents: 0 means they have an identical vector. One very common way to compute the weighted word frequency for word w in a particular document is using $tf - idf$ which stands for term frequency-inverse document frequency. This is defined to be $tf - idf(w) = tf(w) * \log(\frac{N}{df(w)})$ where $tf(w)$ is the frequency of w in the document, N is the total number of documents and $df(w)$ is the number of documents that contain w. Thus the number of occurrences of a word w is weighted by how unusual the word is in the collection. This captures the intuition that less common words provide a better way of distinguishing between documents.

As an example consider the vector based on the words "cat", "dog", "aardvark", "kangaroo" and that these have the following frequencies in three documents:

Document	cat	dog	aardvark	kangaroo
A	3	4	0	0
B	2	3	3	0
C	1	0	0	1

The inverse document frequency for each word is

Document	cat	dog	aardvark	kangaroo
idf	$0 = \log(3/3)$	$0.58 = \log(3/2)$	$1.58 = \log 3$	$1.58 = \log 3$

and so the vector for each document is

Document	cat	dog	aardvark	kangaroo
A	0	0.44	0	0
B	0	1.74	4.74	0

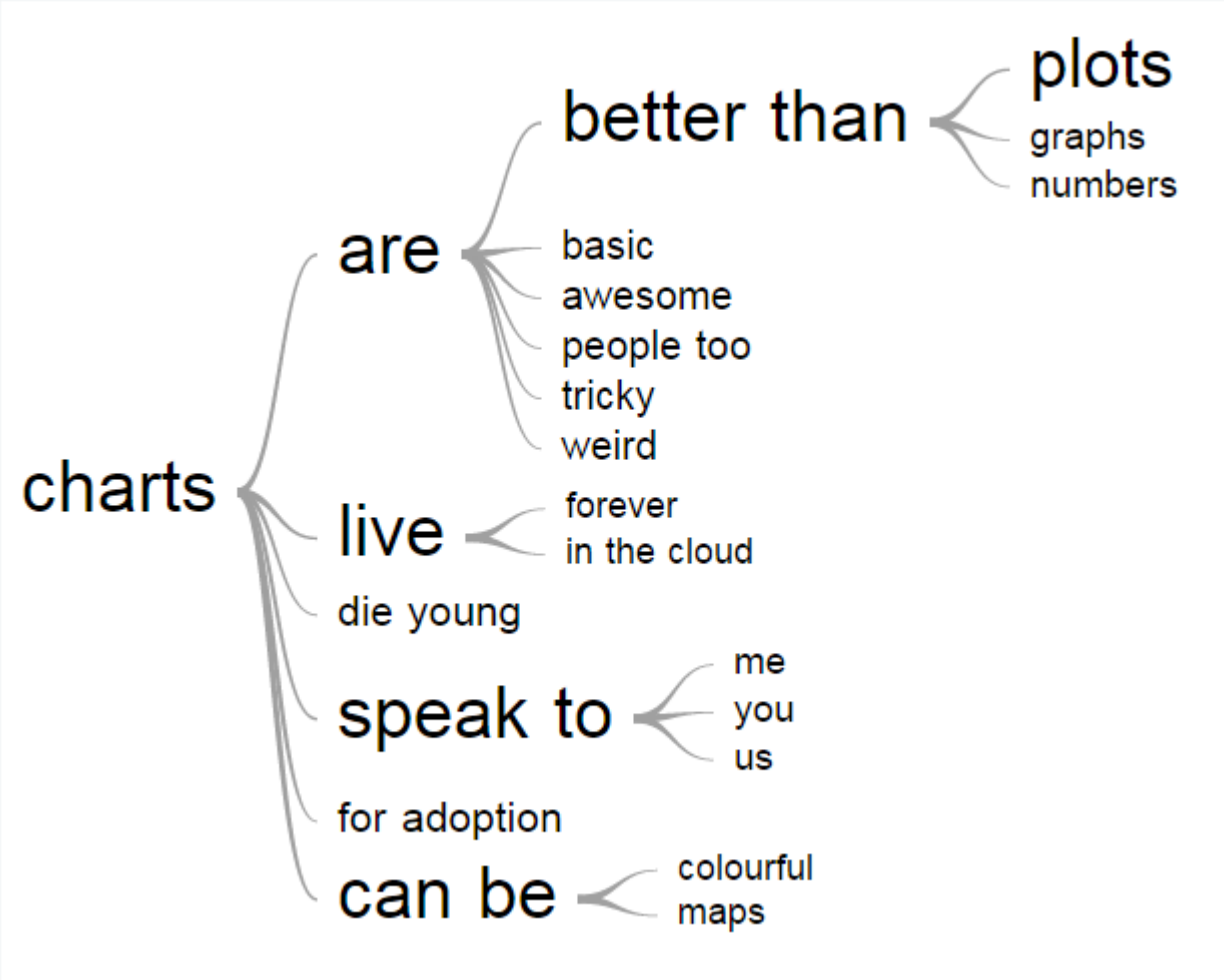


Figure 4.3: Word tree based on: https://developers.google.com/chart/interactive/docs/gallery/wordtree (see: https://jsfiddle.net/hhbw61u4/)

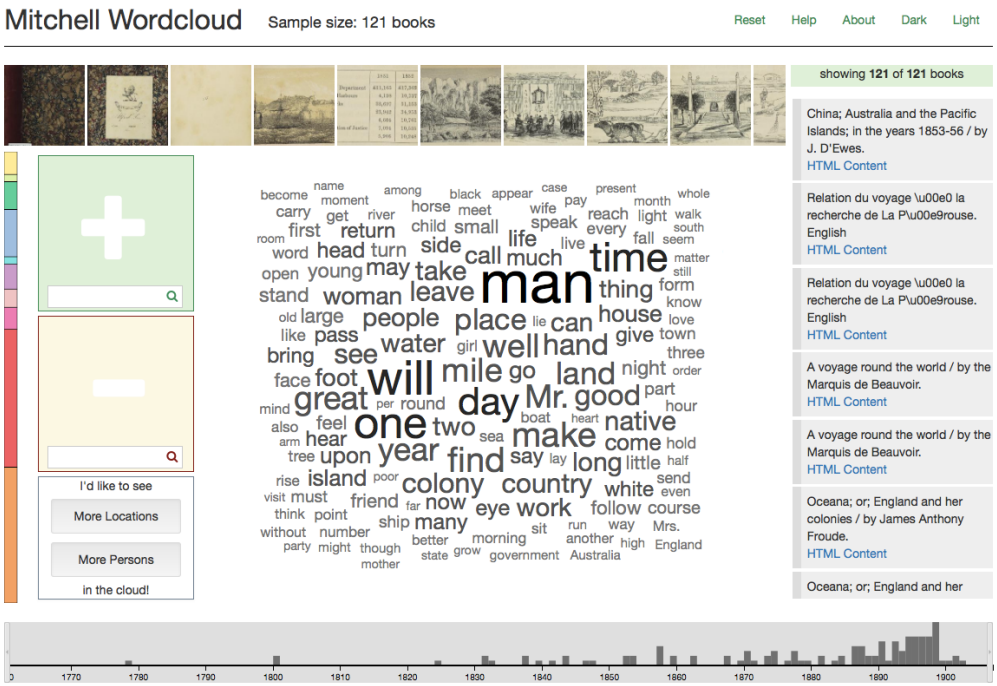


Figure 4.4: License: Copyright © Monash University, unless otherwise stated. All Rights Reserved. The Mitchell WordCloud prototype provides an explorative interface for exploring digitised books in the Mitchell Library collection.

topic a meaningful name if they like, and then analyse the document corpus by identifying the topics that each document covers. Documents can then be clustered or visualised based on topic similarity.

Blei (2012) gives an example of topics that were identified from articles in the Science journal. One topic contained the words “computer”, “models”, “information”, “data”, “computers”, “system”, “network”, “systems”, “model”, “parallel”, “methods”, “networks”, “software.” This collection of words was identified purely on the grounds that they commonly occur near each other, yet it is clear they are semantically related and belong to a meaningful category that a human might label as “computing.”

Topic modelling is also used to look at dynamically changing document corpuses. It can reveal how the choice of topics evolves over time. A stream graph is often used to show these changes.

Document summarisation is also used with collections of documents and can, for instance, be used to summarise all of the documents about a particular topic.

4.4 Explorative Interfaces

One interesting application of interactive data visualisation is for providing on-line interfaces to digital document collections that move beyond traditional Boolean search. Such interfaces should provide an overview of the collection and entice members of the general public to explore the collection, allowing serendipitous discoveries.

In a recent Masters project Monika Schwarz designed such an on-line interface for the David Scott Mitchell collection of the State Library of New South Wales. A central word cloud shows the most prominent words of the collection. Words from the cloud can be dragged into containers to make a positive or negative selection.

The title list to the right shows a ranked list of titles based on the selection criteria and also offers preview word clouds for individual books and access to the digitised book. The image line on top shows the images from the ranked list or an individual book. The time line at the bottom shows the publication years of the books while the bar to the left uses the Dewey classification system to provide orientation within a collection. All the elements of the application are manipulable and linked providing an engaging responsive experience.

4.5 Text analysis tools

There are a wide variety of tools available for text analysis. Unfortunately there is no single best tool as the tools typically only offer only a subset of more advanced text analysis like topic modelling or automatic summarisation. Here is a very partial list:

- Both R and Python have libraries for text analysis. The most widely used R library is tm and in Python it is the powerful Natural Language Toolkit (NTLK).
- SAS Visual Analytics and Tableau both provide simple text analytics.
- The Java-based Stanford NLP tools are widely used.
- There are also a variety of specialised commercial tools.

4.6 Conclusion

Text from social media, blogs, newspaper articles and other online documents provides an amazing source of data for the data scientist. By analysing twitter feeds they can see what are the hot new topics. However textual documents are one of the most difficult kinds of data sources to work with. Here we have only scratched the surface and you are encouraged to go and read more.

FURTHER READING

An interesting place to start is the Text Visualisation Browser which provides an interactive visualisation tool for exploring the collection of papers about text visualisation techniques.

Also see

Ward, Matthew O., Georges Grinstein, and Daniel Keim. Chapter 10 of *Interactive data visualization: foundations, techniques, and applications (2nd Ed).* CRC Press, 2015.

You can read more about topic modelling in

Blei, David M. Probabilistic topic models. *Communications of the ACM 55(4)(2012): 77-84.* Available from http://www.cs.columbia.edu/~blei/papers/Blei2012.pdf

---

Chapter 5

Activity: Text analysis and visualisation with R

By Laurens Skelly, Kimbal Marriott

Updated 17 March 2018

5.1 Text analysis and visualisation with R

Wordclouds are one way to visualise word collections e.g. a document (or collection of collections = corpus).

If your text is not clean & tidy then you may have to transform or 'clean' it.

There are many transformations that can be made to raw text e.g.:

- Convert the text to lower case
- Remove numbers
- Remove common stopwords (e.g. 'the')
- Remove your own stop words (e.g. for a specific topic)
- Remove punctuation
- Remove extra white space
- Text stemming (common root words, not necessarily real words)
- Lemmatisation (common dictionary root words)
- Special characters e.g. replacing "/", "@"
- Really special characters like ü ö ä Ä Ü Ö (and those are just the umlauts)
- Really, really special characters like unprintable whitespace, unicode, emoticons

Based on: http://www.sthda.com/english/wiki/text-mining-and-word-cloud-fundamentals-in-r-5-simple-steps-you-should-k

Data file download: WordCloud.txt

```
# we need the following libraries
library("tm") # for text mining
library("SnowballC") # for text stemming
library("wordcloud")
library("RColorBrewer")

# Read the text file
filePath <- "WordCloud.txt" # or get your own this is one of subject chapters
text <-readLines(filePath) # Load the data as a corpus and save a copy as 'input'
```

```
input <- Corpus(VectorSource(text))
docs <- input
```

You can 'inspect(docs)' but it's messy. Just open the file "WordCloud.txt" to inspect.

Or look at the source (may have changed): https://5147.yalongyang.com/book/module04/exploring-text-and-document-colle
html

Do some processing

We want to isolate all the words, convert to a matrix, count them, sort them, dump them into a DataFrame (for display and for wordclouds)

```
dtm <- TermDocumentMatrix(docs)
m <- as.matrix(dtm)
v <- sort(
  rowSums(m),
  decreasing = TRUE
)
d <- data.frame(
  words = names(v),
  freq = v
)
nrow(d) # how many 'words'
head(d,10) # look at top 10
```

How many 'words' are there?

Have a look at the tail too, not so pretty...

```
tail(d ,10)
# or look at all the words with
# d$word
# generate the raw word cloud, OK here because it's 'reasonably' tidy text
# up to you if you want to try this with e.g. tweetsset.seed(1234)
# so that the cloud is reproducible, optional

wordcloud(
  words = d$words,
  freq = d$freq,
  min.freq = 1,
  max.words = 200,
  random.order = FALSE,
  rot.per = 0.35,
  colors = brewer.pal(8,"Dark2")
)
```

Your turn, change the min.freq then the max.words and run a few times to test

What's the difference between the two parameters?

We can see from the frequency table and the wordcloud, that 'the' is a winner.

Do the common words (e.g. 'the') distort our analysis?

These common words are referred to as 'stop words'

So, before we remove them... who are these stopwords anyway?

```
stopwords("english")
stopwords("french") # c'est la vie...
stopwords("romanian") # why not...
stopwords("swahili") # no... boo

docs <- tm_map(docs, tolower) # make all letters to lowercase
docs <- tm_map(docs, removeWords, stopwords("english"))
# this is destructive, you may need to start again or use the copy 'input'
# and process again

dtm <- TermDocumentMatrix(docs)
m <- as.matrix(dtm)
v <- sort(rowSums(m),decreasing=TRUE)
d <- data.frame(words = names(v), freq=v)
nrow(d)
head(d, 10)

# and display again, 'the' should be gone
set.seed(1234)
wordcloud(
  words = d$words, freq = d$freq, min.freq = 1,
  max.words=200, random.order=FALSE, rot.per=0.35,
  colors=brewer.pal(8, "Dark2")
)
```

use `?TermDocumentMatrix` to have a look at the data you used.

Notice above 'word' and 'words' ('document' & 'documents' etc.).... how to fix that?

```
stemCompletion(c("visualis","techniqu","identifi"), d$words)
stemCompletion(c("visualis","techniqu","identifi"), d$words, type = "prevalent")
stemCompletion(c("visualis","techniqu","identifi"), d$words, type = "random")
```

Also try to run the above code a few times to see what happens.

This is called **stemming**, which is finding the word stem of an input.

Is stemming reversible?

What does 'prevalent' mean/do?

Let's go ahead and stem the doc.

```
docs <- tm_map(docs, stemDocument)
dtm <- TermDocumentMatrix(docs)
m <- as.matrix(dtm)
v <- sort(rowSums(m),decreasing=TRUE)
d <- data.frame(words = names(v),freq=v)
nrow(d)
head(d, 10)
set.seed(1234)
wordcloud(
  words = d$words, freq = d$freq, min.freq = 1,
  max.words=200, random.order=FALSE, rot.per=0.35,
  colors=brewer.pal(8, "Dark2")
)
```

The more fixes you make, the more the anomalies rise from the depths http://stackoverflow.com/questions/

24311561/how-to-use-stemdocument-in-r

Consider also the order in which you perform these transformations (e.g. stemming, stopwords), is any repetition required?

Try stemming first?

Note also that punctuation confuses wordStem so you would have to take them out as well.

http://stackoverflow.com/questions/7263478/snowball-stemmer-only-stems-last-word

Stemming first, input is original data.

```
# stemming first, input is original
docs <- tm_map(input, stemDocument, language = "english")
# docs <- tm_map(docs, stemDocument)
dtm <- TermDocumentMatrix(docs)
m <- as.matrix(dtm)
v <- sort(rowSums(m),decreasing=TRUE)
d <- data.frame(words = names(v),freq=v)
nrow(d)
head(d, 10)
set.seed(1234)
wordcloud(words = d$words, freq = d$freq, min.freq = 1,
          max.words=200, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))


# Remove common English stopwords (e.g. 'the')
docs <- tm_map(docs, removeWords, stopwords("english"))
dtm <- TermDocumentMatrix(docs)
m <- as.matrix(dtm)
v <- sort(rowSums(m),decreasing=TRUE)
d <- data.frame(words = names(v),freq=v)
nrow(d)
head(d, 10)
set.seed(1234)
wordcloud(words = d$words, freq = d$freq, min.freq = 1,
          max.words=200, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))
```

Associations

Which words occur with, or alongside others

```
findAssocs(dtm, terms = "kimbal", corlimit = 1) # ha (and note lowercase, must be automated)
```

What does corlimit mean here?

```
findAssocs(dtm, terms = "text", corlimit = 0.5) #
```

Lemmatizers

Note that lemmas differ from stems in that a lemma is the root form of the word, while a stem may not be a real word.

There are a few Lemmatizers about e.g.

- Stanford NLP

- Treetagger
- koRpus # seems better to install locally…
- Wordnet
- NLTK (Python)

Here's one that works:

http://stackoverflow.com/questions/22993796/lemmatizer-in-r-or-python-am-are-is-berequire

You may find the url is different, it is because the Northwestern University update their URL for this service.

```
require(httr) # may have to install these two
require(XML)

lemmatize <- function(wordlist) {
  get.lemma <- function(word, url) {
    response <- GET(url,query=list(spelling=word,standardize="",
                                   wordClass="",wordClass2="",
                                   corpusConfig="ncf",    # Nineteenth Century Fiction
                                   media="xml"))
    content <- httr::content(response, type="text/plain")
    xml     <- xmlInternalTreeParse(content)
    return(xmlValue(xml["//lemma"][[1]]))
  }

  # here's the code
  url <- "http://classify.at.northwestern.edu/maserver/lemmatizer"
  return(sapply(wordlist, get.lemma, url=url))
}

words <- c("is","am","was","are")
lemmatize(words)
# is am  was  are
# "be" "be" "be" "be"
# 'to be'
words <- c(
  "walk","walking","walked","walk",
  "Walker", "walks", "Walker Texas Ranger",
  "sidewalk"
  )
lemmatize(words)
words <- c("Walker Texas Ranger","range","ranger","ranged","rangy","Ranger")
# so ranger as in rang a bell!? <sigh> English </sigh>
lemmatize(words)
```

5.2 Text networks

We can apply network analysis to text, or more precisely terms or words in text and their connections

e.g. 'data' and 'mining' may commonly occur together

Below is a selection of terms from a document with connections between them represented as a matrix

Data file download: termDocMatrix.rdata

```
# load termDocMatrix
load("termDocMatrix.rdata")
```

```
# inspect part of the matrix
termDocMatrix[5:10,1:20]
# Transform Data into an Adjacency Matrix
# change it to a Boolean matrix
termDocMatrix[termDocMatrix>=1] <- 1
# transform into a term-term adjacency matrix
termMatrix <- termDocMatrix %*% t(termDocMatrix)
# inspect terms numbered 5 to 10
termMatrix[5:10,5:10]
```

Why is 'data' most commonly associated with 'data' i.e. what does 'data' 53 mean?

Inspect the diagonal (top left to bottom right)

Build a graph from the matrix:

```
library(igraph)

g <- graph.adjacency(termMatrix, weighted=T, mode = "undirected")

# remove loops
g <- simplify(g)

# set labels of vertices
V(g)$label <- V(g)$name

# set seed to make the layout reproducible
set.seed(3952)
layout1 <- layout.fruchterman.reingold(g)
plot(g, layout=layout1)
# plot(g, layout=layout.kamada.kawai)
```

How many times did 'r' occur in the text?

Try other layouts.

Which is better to learn about relationships between data, the matrix or the graph?

Chapter 6

Activity: Textual Exploration with Voyant Tools

By Monika Schwarz

Updated 17 April 2018

6.1 What is Voyant Tools?

Voyant Tools has been designed as a web-based text reading and analysis environment. It is a scholarly project created by Stéfan Sinclair and Geoffrey Rockwell which offers visual analysis technics originally aiming at digital humanities students and scholars. Over the last few years Voyant Tools has matured into a sophisticated tool that includes an array of different text visualisations ranging from very standard to extremely experimental. We are going to use Voyant tools to look at some of them.

6.2 What can I do with it?

With Voyant Tools you can both explore a single text document or an entire corpus of text documents. You can easily create single stand-alone visualisations or combine them into multiple interactive views side-by-side. By default, such an interface is shown when a corpus or document is first loaded but there are many ways to tweak and customize it to your liking. And finally, there are options to export a link of your work as an HTML snippet or an image to incorporate in your own webpages and texts.

6.3 How do I start?

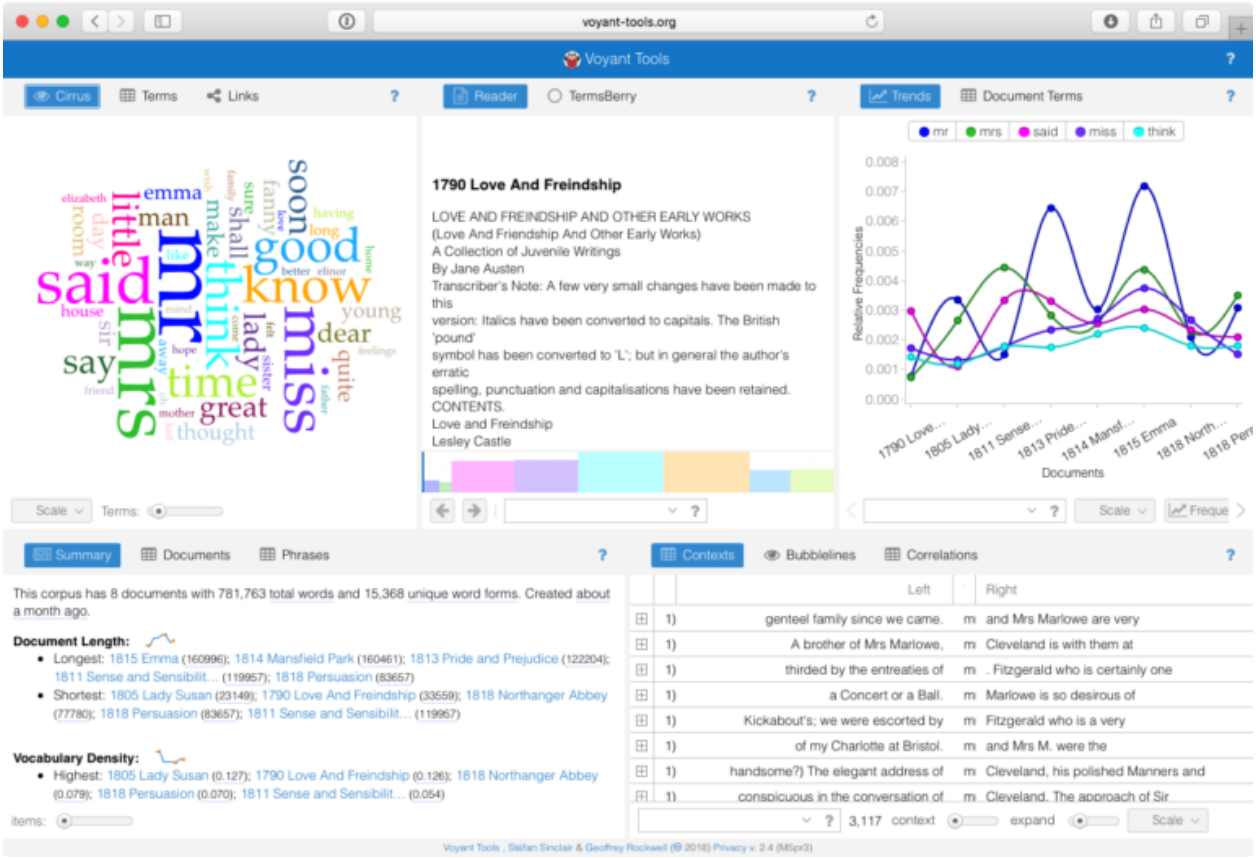
Access to Voyant tools is easy. You simply load a text or an entire text corpus on the start page. Voyant tools accepts text in various formats like plain text, HTML, PDF or MS word. You can also load a collection of text using a zip file or a set of URLs. We are going to use an already existing corpus for our first exploration and upload our own text for the second exercise.

6.4 Exercises

6.4.1 Exploring a corpus of documents

Go to Voyant Tools (http://voyant-tools.org)

Click the 'Open' button, a dialog should appear. Choose the corpus 'Austen's Novels' and open. Voyant now fetches the corpus and automatically opens the main interface. Under the surface Voyant does a lot of pre-processing for us like removing punctuation, excluding stopwords and building a term-document matrix as a basis for the visualisations.



Voyant offers about 25 different views and by default the interface shows five of them, which can be switched to different views via tabs. On the top left you see a word cloud which in Voyant is called Cirrus (Latin for 'cloud') showing the most common terms. This visualisation uses Jason Davies' D3-based Word Cloud library, which itself is an implementation of Wordle.

Next to Cirrus the Reader view shows the actual text. The Trends view shows the distribution of the top 5 terms over the entire corpus. The Cirrus and Trends views are interactive.

- Try it out: In the word cloud click on names like Elinor, Elizabeth or Emma to find out in which novel these characters are the heroine. Has Jane Austen used these names only once or has she recycled some of them in other books?
- Try it out: Look for the word 'love' in the word cloud (or type it into the search box in Trends) and compare the relative and the raw frequency in Terms. Why are the resulting curves so different?

At the bottom left you find a summary of the entire corpus, which consists of eight still highly popular novels written by Jane Austen. You see statistical details on the document length, vocabulary density or average words per sentence.

- Look at distinctive words: These are words that appear only in one novel and are therefore within this corpus typical for this particular novel. What do you think makes these words so unique?

The last view on the bottom right shows terms in context. More specifically it shows a term in the centre as well as the five words before and after the term.

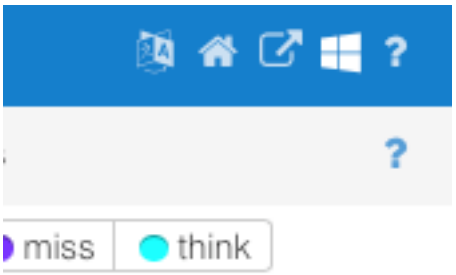
- Try typing your own terms into the search box to look at them in context.

- For fans of the 1995 TV adaptation of 'Pride and Prejudice': Can you find the scene where Mr. Darcy (played by Colin Firth) confesses his love to Elizabeth? (Tip: try looking for 'ardently', then expand the content by pressing the + sign.)

Beside these first five views Voyant Tools offers about 20 more. You can find a full list and description of tools in the official guide: http://voyant-tools.org/docs/#!/guide/tools. Some of them are easily accessed via tabs, e.g. the Reader View has a TermsBerry view on the second tab. The TermsBerry actually works from a similar angle as Contexts. It uses the proximity of terms to establish connections between them. The outcome is much more playful. By default, the most common terms are shown. When you hover over a term the words that appear most often in its context will be highlighted.

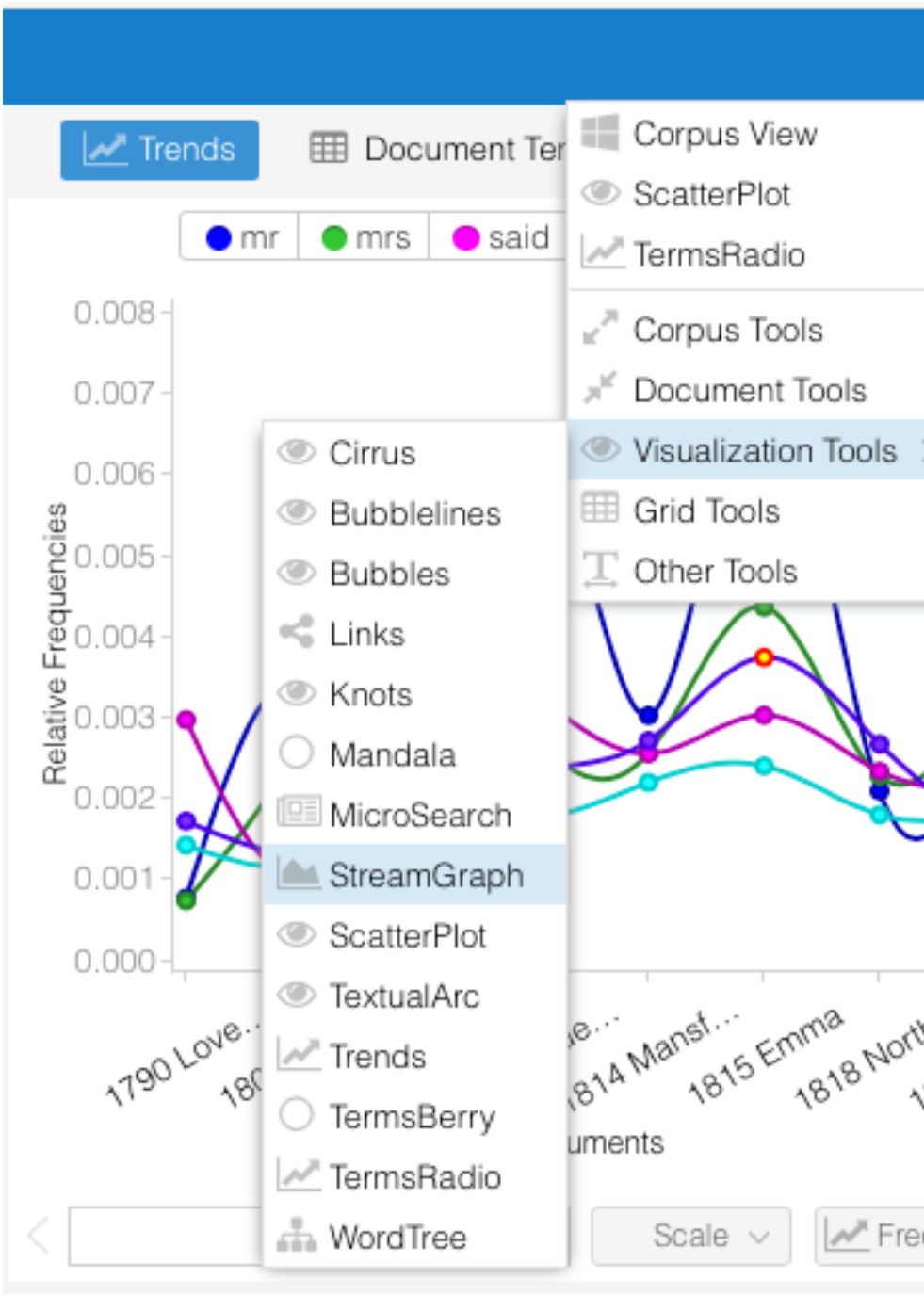
- Try it out: Hover over Marianne and Jane to find the name of their respective sisters. By default, the context is set to the next two words. Try expanding it to five or more.
- Try it out: Change the strategy to see the distinct terms. Hovering over a term gives us the term frequency and, relevant for distinct terms, its TF-IDF score. The formula for TF-IDF is used here to find the terms that have a high frequency in the corpus but mostly occur in one particular document. Like we saw before, within the corpus the words truly unique to one book would be the names of its characters.
- The size of the circles indicates the number of times a term is mentioned. Can you guess at the names of the top five female heroines in the Austen novels?

You can access the full range of views via the question mark on the top right of each view or to create a single visualisation, on the top right of the screen. Hover over it, move to the window and choose a view from the dropdown menu. The views above the lines are the ones Voyant suggests as the most helpful ones. Via the question mark you can also set options like a different stopword list or a specific color palette and font for the word cloud.



Now let's create a StreamGraph. Stream graphs are usually used for showing occurrences of terms or topics over time (like Twitter streams) but it can also show the distribution over a corpus of novels ordered by their publication years.

- Use the question mark on the top right to go to Visualisation Tools / StreamGraph. (You can always go

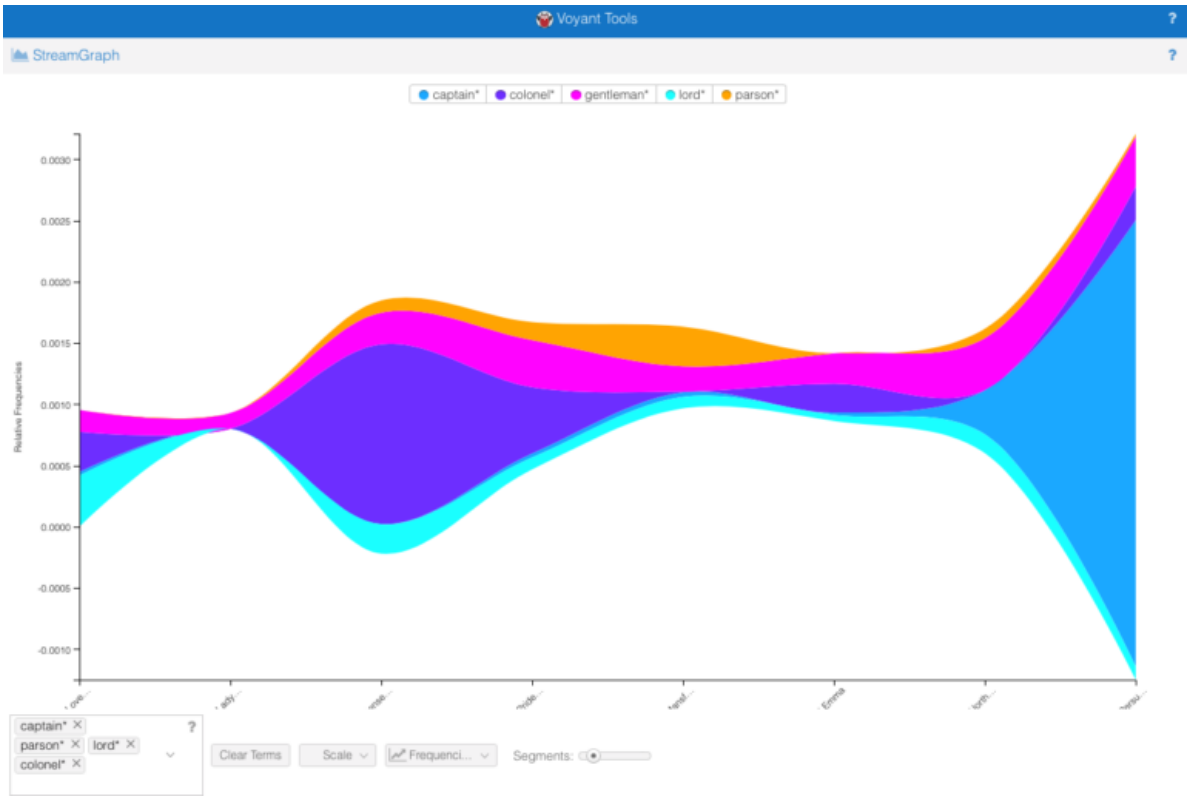


back by clicking `'Corpus View'`).

- Since the most common terms are a bit to general let's have a look at somewhat more specific terms. Austen's novels are all set in the world of the British gentry around 1800 and a male character would typically be a gentleman, but some thrived to become something more specific.

- Delete all terms via the `'Clear terms'` button and then enter the term 'gentleman' into the search box – you have to type it first and then once loaded choose the term from a dropdown list.

- Now add `'colonel'`, `'lord'`, `'parson'` and `'captain'` in the same way. Some terms are evenly distributed, but some professions (or titles) seem to be much more relevant in some novels than in others.



- What about lower professions? Enter `'sailor'`, `'soldier'` and `'farmer'`. Nah, they really weren't part of Austen's world.

Now let's look at the export options. You can simply export the URL of your visualisation. You can also export an image as a PNG or SVG file or a HTML snippet to include in your own webpage. We'll try the last one.

- Go to the question mark on the top left corner and then click the export button. Choose `'Export View'` and click `'an HTML snippet for embedding this view in another web page'`.
- Download the index3.html file.
- Open the index3.html file in an editor of your choice (e.g. Atom or Brackets).
- Paste the iframe into the body of the html document.
- Since you are opening the html document as a local web page you need to specify the source. Add `'https:'` at the beginning of the source tag (src='https://voyant-tools.org/tool/……'). You can also adjust the width and height.
- Save it and open the html document in your browser. Sometimes it takes a little while to load the image.

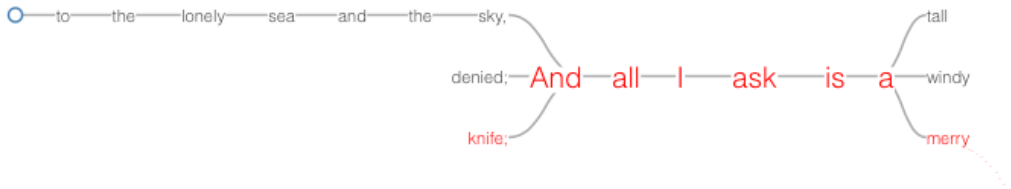
6.4.2 Exploring a single text document

Now let's see what Voyant can do with just a single document.

- Download the document, but do not yet read it!
- Open a new tab on your browser and go to Voyant tools (http://voyant-tools.org) again.
- This time click 'Upload', find the 'Seafever' text on your local machine and open it. Click the TermsBerry to avoid reading the text. We will use the different views instead to get an impression of the text.
- Have a look at the word cloud first. Do you get an idea of what this text might be about?
- Play around with the TermsBerry again, does the impression change? Also note, with a single text certain corpus-related possibilities like the distinct term strategy are not possible unless you subdivide

the text into chapters. Our text is too short for that.

- Switch from Trends to Document Terms. Here you'll find a couple of words showing up three or two times but most terms are only mentioned once. The sparklines indicate the position of that term in the text.
- The Contexts view shows the most frequent term 'ask'. The left side shows it is actually part of the phrase 'And all I ask'.
- Let's try a few new things: switch the Summary view to Phrases. This shows us text repetitions. Next to the one we already discovered the even longer 'I must go down to the seas again…' is used twice. Can you guess at the nature of the text now? Is this a code book? An IKEA instruction manual? Or did we maybe land in the strange of poetry?
- Now switch one of the views to Visualisation Tools / Wordtree. Click on the terms to extend the lines and explore the context of the 'And all I ask' sequence.



Now have a look at the text. It is indeed a poem. Seafever is part of the Salt-Water Poems and Ballads of John Masefield (1878-1967) (The Trekkies amongst might appreciate this video clip where Captain James T. Kirk cites the first line and explains it's meaning. https://www.youtube.com/watch?v=-eXB1Yj05Fw). Poetry often expresses a feeling or a mood, in this case a longing for travelling the seas, through words and no visualisation could ever fully capture this. But judge for yourself: did you maybe indeed get a certain idea of the meaning of this poem without actually reading the text first?

6.4.3 (Optional) Additional Exercises

Have a look at the somewhat more experimental stuff.

- Go back to the Austen corpus we looked at in the first exercise. Use the window icon > 'Vizualisation Tools' to open TextualArc. (Note: depending on browser and connection this might be slow to load). This visualisation is inspired by W. Bradford Paley's ground-breaking work TextArc (for more see http://vallandingham.me/textarc/). The entire corpus text is once ordered clockwise around the perimeter of a circle, starting from the top. Inside the circle the most prominent terms are placed depending on their positions in the text. The text is displayed running through a single line underneath the circle. While it reads a second line traces the words in reading order within the circle.
- Open Knots to see single terms traced by lines. Every time a term occurs in the corpus the line takes a little turn to the right creating twists.
- Have a conversation with Veliza. This is the most experimental view on Voyant offering a (somewhat limited) natural language exchange in the style of a therapy session. This work is inspired by the Eliza program written by Joseph Weizenbaum in the 1960s (https://en.wikipedia.org/wiki/ELIZA).