Text Mining and Sentiment Analysis

Prof. Annamaria Bianchi

A.Y. 2024/2025

Lecture 8

11 March 2025

UNIVERSITÀ DEGLI STUDI DI BERGAMO
Dipartimento di Scienze Economiche

Outline

Frequency plot

Word cloud

Word association

Word networks

Packages: **ggplot2, wordcloud, widyr, igraph, ggraph**

Functions: `wordcloud::wordcloud(), widyr:: pairwise_count(), widyr::pairwie_cor(), igraph::graph_from_data_frame(),ggraph::ggraph(),`

`dplyr::slice_min(), slice_max()`

Bar charts

A bar plot is the simplest graph to display term frequencies. There are two types of bar charts: `geom_bar()` and `geom_col()`:

- `geom_bar()` makes the height of the bar proportional to the number of cases in each group (or if the weight aesthetic is supplied, the sum of the weights). `geom_bar()` uses `stat_count()` by default: it counts the number of cases at each x position.
- `geom_col()` makes the height of the bar proportional to values in the data. `geom_col()` uses `stat_identity`: it leaves the data as is.

`coord_flip()` switches the x- and y-axes. This is useful for example if you want horizontal bars and in case of long labels.

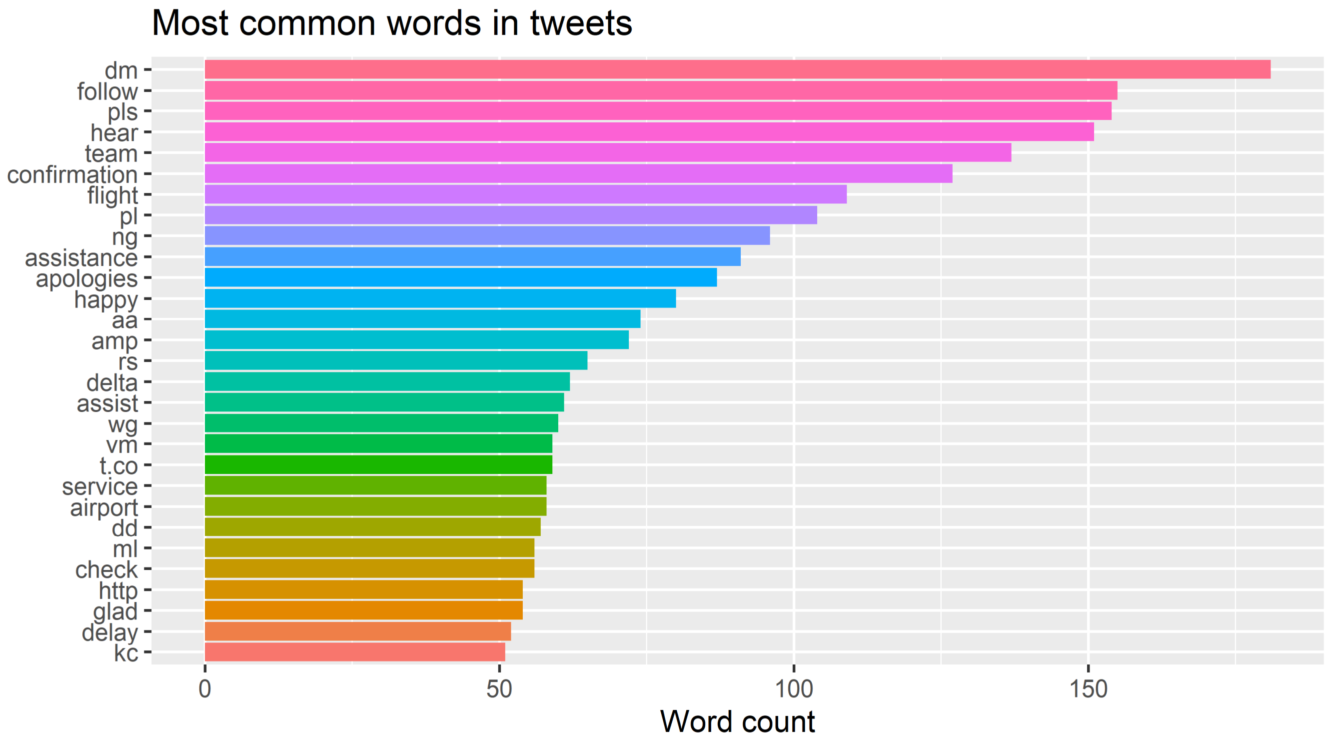
To control the order of the terms we use the `reorder()` function.

Term frequency visualization

Let us start from the freq.df.4 data frame object. This becomes the data used by ggplot2 to construct the bar plot

```
> freq.df |> filter(n>50) |>
+ mutate(word = reorder(word, n)) |>
+ ggplot(aes(word, n, fill = word)) +
+ geom_col(show.legend = F) +
+ xlab(NULL) +
+ ylab('Word count') +
+ ggtitle('Most common words in tweets') +
+ coord_flip()
```

Term frequency visualization



Word cloud

Another common visualization is called a **word cloud**. A word cloud (or tag cloud) can be an handy tool when you need to highlight the most commonly cited words in a text using a quick visualization.
Generally, a word cloud is a visualization based on frequency. In a word cloud, words are represented with **varying font size**.
In a simple word cloud, only one dimension of information is shown. Specifically, the font size corresponds to word frequency. This means that the larger a word in the word cloud, the more frequent the word is in the corpus.
Other dimensions of a word cloud can be changed to demonstrate new information, such as color and grouping.

In general, word clouds are popular because audiences can easily comprehend the illustration. This has led to an over use of word clouds during text mining projects. In general, it is best to use word clouds sparingly despite their popularity.

We will use the **wordcloud** library. It has several interesting wordcloud functions. The simplest is named `wordcloud`

Word cloud

`wordcloud()`	Plot a word cloud

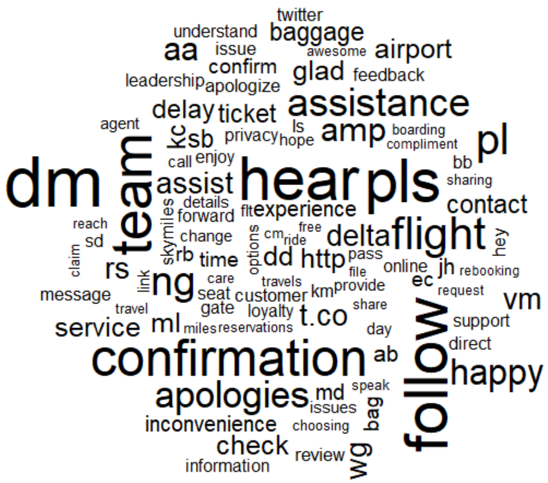
```
wordcloud(words,freq,scale=c(4,.5),min.freq=3,max.words=Inf,
      random.order=TRUE, random.color=FALSE, rot.per=.1,colors="black", ...)
```

`words`	the words
`freq`	their frequencies
`scale`	A vector of length 2 indicating the range of the size of the words.
`min.freq`	words with frequency below min.freq will not be plotted
`max.words`	Maximum number of words to be plotted. least frequent terms dropped
`random.order`	plot words in random order. If false, they will be plotted in decreasing frequency
`random.color`	choose colors randomly from the colors. If false, the color is chosen based on the frequency
`rot.per`	proportion words with 90 degree rotation
`colors`	color words from least to most frequent

Word cloud

```
> install.packages("wordcloud")
> library(wordcloud)

> dev.new(width = 3000, height = 1500, unit = "px")
NULL
> wordcloud(freq.df$word,
+ freq.df$n,
+ max.words=100)
```



Word cloud

```
> wordcloud(freq.df$word, freq.df$n,
+ max.words=100,
+ rot.per=.3,
+ colors=brewer.pal(8,"Dark2"))
```


```
> brewer.pal.info
> display.brewer.pal(8, "Dark2")
```

Word associations

In text mining, association is similar to correlation. That is, when term x appears, the other term y is associated with it. It is not directly related to frequency, but instead refers to the term pairings.

We may want to understand which pairs of words co-appear.

To this purpose, we use the package **widyr**

```
> install.packages("widyr")
> library(widyr)
```

With reference to the DeltaAssist example, we want to explore the word associations with the term «apologies». The term «apologies» was chosen after first reviewing the frequent terms for unexpected items, or in this case, to learn about a behaviour of customer service agents.

Word associations

First, we count common pairs of words co-appearing within the same tweet

`widyr::pairwise_count()`	Count pairs of items within a group

```
pairwise_count(tbl, item, feature, ...)
```

`tbl`	Table
`item`	Item to count pairs of; will end up in item1 and item2 columns
`feature`	Column within which to count pairs item2 columns

Word associations

```
> word_pairs = tidy.tweets.2 |>
+ pairwise_count(word, ID, sort = TRUE)
> word_pairs
# A tibble: 39,734 × 3
   item1        item2            n
   <chr>        <chr>        <dbl>
 1 dm           follow         129
 2 follow       dm             129
 3 confirmation dm              93
 4 dm           confirmation    93
 5 confirmation follow          78
 6 follow       confirmation    78
 7 follow       pls             59
 8 pls          follow          59
 9 t.co         http            54
10 http         t.co            54
# … 39,724 more rows
> View(word_pairs)
```

The output provides the pairs of words as two variables (item1 and item2). This allows us to perform normal text mining activities like looking for what words are most associated with "apologies"

Word associations

```
> apol_pairs = word_pairs |>
+ filter(item1 == "apologies") |>
+ arrange(desc(n))
> View(apol_pairs)
```

	item1	item2	n
1	apologies	delay	22
2	apologies	pls	14
3	apologies	issues	12
4	apologies	follow	10
5	apologies	dm	10
6	apologies	confirmation	10
7	apologies	assistance	9
8	apologies	inconvenience	8
9	apologies	hear	8

Word associations

The most common co-appearing words only tells us part of the story. We may also want to know how often words appear together relative to how often they appear separately, or the *correlation* among words. Regarding text, correlation among words is measured in a binary form - either the words appear together or they do not. A common measure for such binary correlation is the **phi coefficient**.

The phi coefficient focuses on how much more likely it is that either both words X and Y appear, or neither do, than that one appears without the other.

Consider the following table:

	Has word Y	**No word Y**	**Total**
Has word X	n_{11}	n_{10}	$n_{1\cdot}$
No word X	n_{01}	n_{00}	$n_{0\cdot}$
Total	$n_{\cdot 1}$	$n_{\cdot 0}$	n

For example, n_{11} represents the number of documents where both word *X* and word *Y* appear, n_{00} the number where neither appears, and n_{10} and n_{01} the cases where one appears without the other.

Word associations

The phi coefficient is:

$$\phi = \frac{n_{11}n_{00} - n_{10}n_{01}}{\sqrt{n_{1\cdot}n_{0\cdot}n_{\cdot 0}n_{\cdot 1}}}$$

Two binary variables are considered positively associated if most of the data falls along the diagonal cells. In contrast, two binary variables are considered negatively associated if most of the data falls off the diagonal.

The `pairwise_cor()` function in **widyr** lets us find the correlation between words based on how often they appear in the same section. Its syntax is similar to `pairwise_count()`.

Word associations

```
> word_cor = tidy.tweets.2 |>
+ pairwise_cor(word, ID) |>
+ filter(correlation>0.11)
> apol_cor = word_cor |>
+ filter(item1 == "apologies") |>
+ arrange(desc(correlation))
> View(apol_cor)
```

	item1	item2	correlation
1	apologies	delay	0.3001660
2	apologies	issues	0.2453281
3	apologies	appears	0.1522548
4	apologies	kitmoni	0.1467906
5	apologies	youâ	0.1467906
6	apologies	deepest	0.1467906
7	apologies	late	0.1330503
8	apologies	strive	0.1157947
9	apologies	refund	0.1157947
10	apologies	joy	0.1157947
11	apologies	diligently	0.1157947

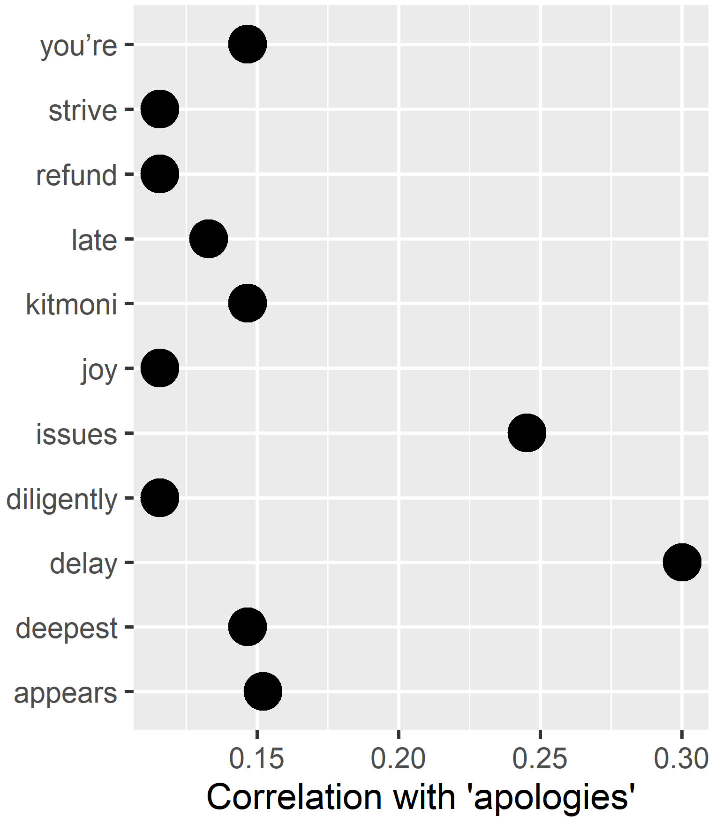
Word associations

Once you have a data frame of highly associated words and their corresponding values, you can use it for building another graph, using ggplot as follows.
Set the y axis to be the terms and the x axis to be the values and use the function `geom_point()`, setting the size explicitly.

We can now produce a plot showing the most associated words with «apologies».

```
> apol_cor |>
+ ggplot(aes(x=correlation, y=item2))+
+ geom_point(size = 4)+
+ ylab(NULL)+
+ xlab("Correlation with 'apologies'")
```

Word associations



The most associated word from DeltaAssist's use of apologies is «delay»

Word Networks

We might be interested in visualizing all of the relationships among words simultaneously, rather than just the top few at a time. A common visualization technique consists in arranging the words into a **network**. Network structures are interesting in conveying multiple types of information visually:

- Used to identify key terms
- Show relationship strength, leading to an assumption of a topic

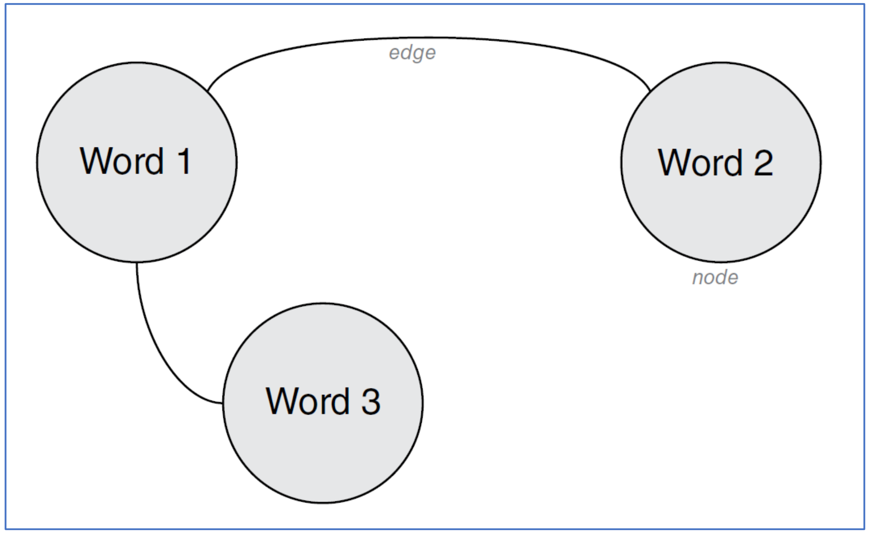
Caution. Word networks can become dense and hard to interpret visually. It is thus important to **restrict** the number of terms that are being connected.

Word Networks

The circles are called **nodes** (or vertices).
The lines connecting the circles are called **edges**.

A network graph can have **many dimensions of information** contained in it. The example presented has the same size nodes and edge thickness. However, some of the parameters can be adjusted:

- size of the nodes showing more prominent members
- thickness of lines representing the strength of the connection
- color denoting particular class attribution (e.g. gender)



Word networks can also be used to understand word choice by visually producing clusters.

Word Networks

A graph can be constructed from a tidy object as long as it has three variables:

- **from**: the node an edge is coming from
- **to**: the node an edge is going toward
- **weight**: a numeric value associated with each edge

In order to produce word networks we will use:

- **igraph** package, which has many powerful functions for manipulating and analyzing networks. We will use it to create an igraph graph object
- **ggraph** package, that implements visualizations using the grammar of graphics

```
> install.packages("igraph")
> library(igraph)
> install.packages("ggraph")
> library(ggraph)
```

Word Networks

In the following, we refer to the Delta Airlines example. We have just seen that the words «apologies» and «refund» are highly associated. A word network may more broadly indicate under what circumstances Delta would issue a refund. We limit the network illustration to the word «refund».

We proceed as follows:

1) Select original tweets containing the word «refund» [we obtain only 7 tweets]
2) To further reduce clutter, we select the first three of the refund-mentioning tweets
3) Transform the text in tidy format and clean it
4) Build a pairwise count data frame
5) Use a function in **igraph** to build an igraph object
6) Build the word network using the package **ggraph**

```
> tweets = read_rds("tweets.rds")
```

Word Networks

1) Select original tweets containing the word «refund» [we obtain only 7 tweets]

```
> refund = tweets |>
+ filter(str_detect(text, regex("refund", ignore_case = T)))
> refund
# A tibble: 7 × 7
weekday month date year text                          agents ID
<chr> <chr> <dbl> <dbl> <chr> <chr> <int>
1 Thu     Oct     1 2015  @lanaandlovely For future r…  KC     49
2 Sun     Oct     4 2015  @gsstan Hello Andrew. Apolog… /2     347
3 Tue     Oct     6 2015  @NickRogersRx I'm sorry, but… WG     487
4 Tue     Oct     6 2015  @NickRogersRx I don't see a … WG     489
5 Sun     Oct    11 2015  @Aj_Marshall17 AJ. Are you a… /2     1004
6 Mon     Oct    12 2015  @Kyrrie_Twin Kyrrie, we offe… VM     1043
7 Mon     Oct    12 2015  @TchCzarina The miles would … EC     1091
```

Word Networks

2) To further reduce clutter, we select the last three of the refund-mentioning tweets
3) Transform the text in tidy format and clean it

```
tidy.refund = refund |>
+ slice_min(ID, n = 3) |>
+ unnest_tokens(word, text) |>
+ anti_join(stop_words) |>
+ filter(!str_detect(word, "\\d"))
Joining with `by = join_by(word)`
> tidy.refund
# A tibble: 22 × 7
weekday month date year agents ID word
<chr> <chr> <dbl> <dbl> <chr> <int> <chr>
1 Thu     Oct   1    2015   KC    49 lanaandlovely
2 Thu     Oct   1    2015   KC    49 future
3 Thu     Oct   1    2015   KC    49 reference
4 Thu     Oct   1    2015   KC    49 fare
```

Word Networks

The functions **dplyr::**`slice_min()` and **dplyr::**`slice_max()` select rows with lowest or highest values of a variable.

```
slice_min(data, order_by, n, ….)
```

`data`	A data frame, data frame extension (e.g. a tibble)
`order_by`	Variable to order by.
`n`	the number of rows to select

Word Networks

4) Build a pairwise count data frame

```
> refund_pairs = tidy.refund |>
+ pairwise_count(word, ID, sort = TRUE)
> refund_pairs
# A tibble: 140 × 3
   item1          item2            n
   <chr>          <chr>        <dbl>
 1 refund         apologies        2
 2 apologies      refund           2
 3 future         lanaandlovely    1
 4 reference      lanaandlovely    1
 5 fare           lanaandlovely    1
 6 options        lanaandlovely    1
 7 refundable     lanaandlovely    1
 8 changeable     lanaandlovely    1
 9 kc             lanaandlovely    1
10 lanaandlovely  future           1
# … 130 more rows
```

Word Networks

5) Use a function in **igraph** to build an igraph object

The function **igraph**::`graph_from_data_frame()` creates igraph graph objects from a data frame

```
graph_from_data_frame(d)
```

`d` A data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes.

```
> refund_network = refund_pairs |>
+ graph_from_data_frame()
> refund_network
```

```
IGRAPH ad46b1a DN-- 20 140 --
+ attr: name (v/c), n (e/n)
+ edges from ad46b1a (vertex names):
 [1] refund       ->apologies     apologies    ->refund
 [3] future       ->lanaandlovely reference    ->lanaandlovely
 [5] fare         ->lanaandlovely options      ->lanaandlovely
 [7] refundable   ->lanaandlovely changeable   ->lanaandlovely
 [9] kc           ->lanaandlovely lanaandlovely->future
[11] reference    ->future        fare         ->future
```

Word Networks

6) Build the word network using the package **ggraph**

We can convert igraph object into a graph with the **ggraph**::`ggraph` function, after which we add layers to it. For a basic graph, we need to add three layers: nodes, edges, and text

```
ggraph(graph, layout = "auto", ...)
```

`graph`	The object containing the graph.
`layout`	The type of layout to create.

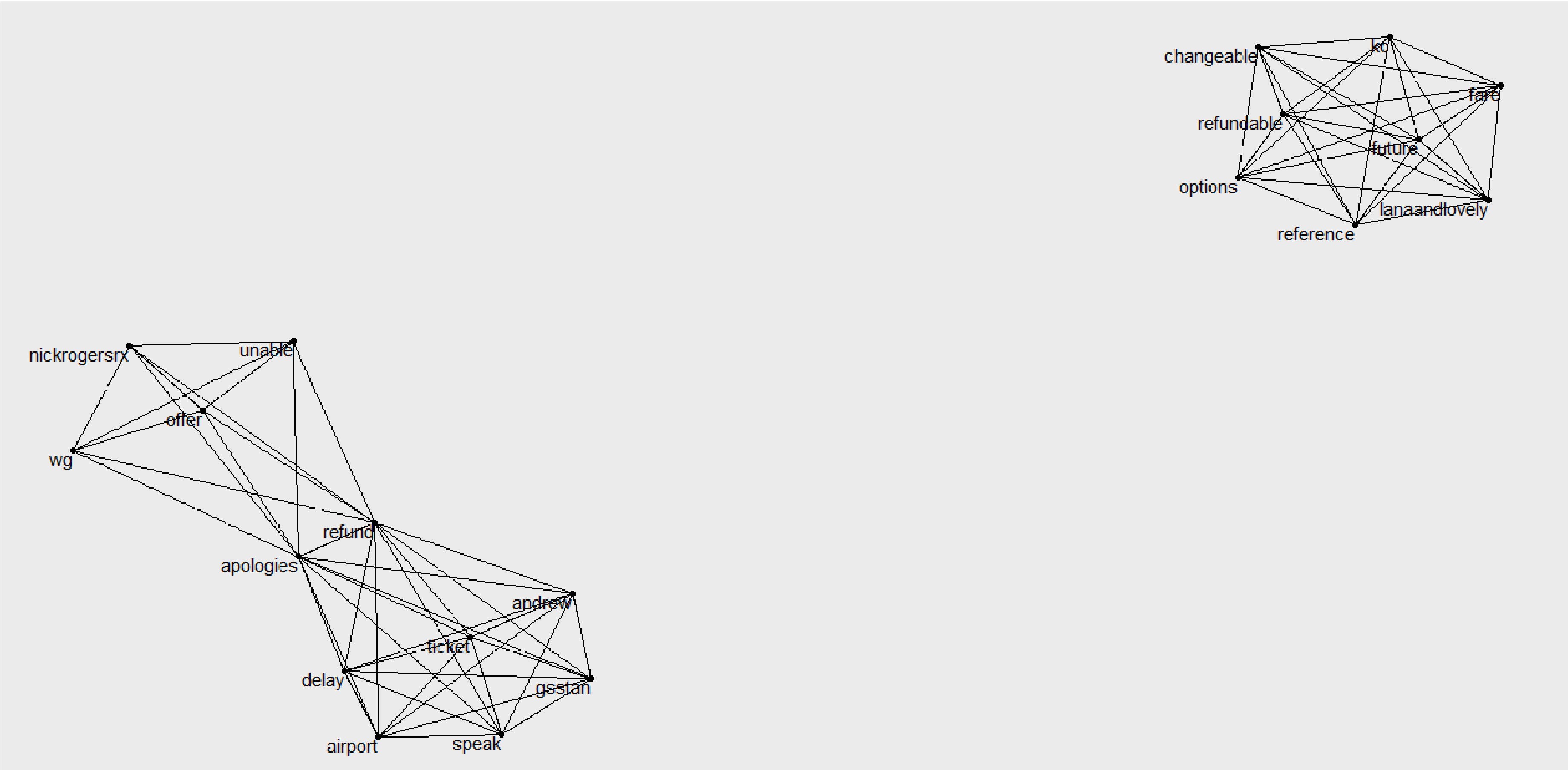
The function **ggraph**::`geom_edge_link()` draw edges as straight lines between nodes
The function **ggraph**::`geom_node_point()` allows for simple plotting of nodes in different shapes, colours and sizes.
The function **ggraph**::`geom_node_text()` annotates nodes with text

Word Networks

6) Build the word network using the package **ggraph**

```
> set.seed(2021)
> dev.new(width = 3000, height = 1500, unit = "px")
NULL
> ggraph(refund_network, layout = "fr") +
+ geom_edge_link() + + geom_node_point() +
+ geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```

Word Networks



Word Networks

The plot allows to visualise some details of the text structure. It shows a strong connection between refund and apologies. We shall identify three clusters, corresponding to the three tweets.

The first two clusters are linked by the words «apologies» and «refund».
Still the third tweet stands alone. This is because it has the word refundable, which was included by the selection, even though it is technically a different term than «refund», so no network connection was created linking all three.

Exercise for you

The data set chardonnay.csv contains tweets related to wine Chardonnay. Write R code to perform the following

1. Import the dataset and create a tibble named chardonnay.tweets.
2. Inspect the imported dataset.
3. Select the variable n.doc and text.
4. Convert the tibble to the tidy format and remove stopwords, creating a new tibble named tidy.chardonnay.
5. Produce the frequency table of words in tidy.chardonnay, named chardonnay.freq.
6. Create a wordcloud for the values in chardonnay.freq. What do you notice?
7. Create a custom stopwords tibble by adding to the tidy dataset stop_words the words "http", "https", "rt", "t.co", "ed“, "amp", "chardonnay", "wine", "glass“.
8. Create a new tibble named tidy.chardonnay.2 by removing the custom stopwords. Further remove all words starting with "00“ and the elements made by one digit.
9. Produce the frequency table of words in tidy.chardonnay.2, named chardonnay.freq.2.
10. Create a wordcloud for the values in chardonnay.freq.2.
11. Explore the use of different colors for the plot. You can take a look at some available colors with `head(colors(),50)`.
12. Explore the use of prebuilt color palettes, using the function `brewer.pal()`.

Exercise for you

13. Build a frequency plot for the most frequent words in the dataset.
14. Explore word co-appearence using the function `pairwise_count()`. Which are the words that co-appear most often? Does that make sense?
15. Explore which are the words most often co-appearing with “cabernet“.
16. Compute the phi coefficient for words co-appearing with “cabernet“.