FIT3152 Data analytics

Assignment 3

Name: Lim Yu-Shan

Student ID: 32685467

Notes to marker:

- The main body of this report is just over 14 pages. Most of this length can be attributed to long code blocks and the sizes of the network graphs that were made big for better readability. All other pages are the Appendix, which includes outputs for network graph statistics such as closeness, betweenness, etc.

Task 1

For this assignment, I collected a set of 23 documents of varying topics from various sources. These documents include poems, song lyrics, famous speeches, news articles, and more. The reference list for these documents is included in the Appendix.

The contents of the documents were copied and pasted into text files.

Task 2

The document contents were already pasted into text files in Task 1. To create the corpus, the text files were placed into a single folder, called `docs`, in the working directory. The following coded is then run, which sets the seed (for reproducibility of graphs), imports the required libraries, and creates the corpus using `Corpus()` from the `tm` package.

```
rm(list = ls())
set.seed(32685467)
library(tm)
library(cluster)
library(igraph)

cname <- file.path(".", "docs")
docs <- Corpus(DirSource((cname)))
list.files("docs")
```

```
##  [1] "asoiaf01.txt"    "asoiaf02.txt"    "asoiaf03.txt"    "bitcoin01.txt"
##  [5] "bitcoin02.txt"   "churchill01.txt" "churchill02.txt" "covid01.txt"
##  [9] "covid02.txt"     "covid03.txt"     "hamlet01.txt"    "hamlet02.txt"
## [13] "linux01.txt"     "linux02.txt"     "peaky01.txt"     "queen01.txt"
## [17] "queen02.txt"     "stats01.txt"     "stats02.txt"     "stats03.txt"
## [21] "yeats01.txt"     "yeats02.txt"     "yeats03.txt"
```

Each document is named based on its source or topic, followed by a number. For example, the contents of `hamlet01.txt` and `hamlet02.txt` are extracts from the same play, *Hamlet*, by William Shakespeare.

Task 3

Before creating the document-term matrix (DTM) from the corpus, some transformations are performed on the texts to get more meaningful tokens out of them. These are shown and explained in the code block below.

```
# function to replace characters with spaces
to_space <- content_transformer(function(x, pattern) gsub(pattern, " ", x))
docs <- tm_map(docs, to_space, "-")                    # replace dashes with spaces
docs <- tm_map(docs, to_space, "\n")                   # replace line breaks with spaces
docs <- tm_map(docs, removeNumbers)                    # remove numbers
```

```
docs <- tm_map(docs, removePunctuation)                    # remove punctuation
docs <- tm_map(docs, content_transformer(tolower))         # convert characters to lowercase
docs <- tm_map(docs, stripWhitespace)                      # remove extra whitespaces
docs <- tm_map(docs, removeWords, stopwords("english"))    # remove English stopwords
docs <- tm_map(docs, stemDocument, language = "english") # stem words
```

The corpus is then converted to a document-term matrix using `DocumentTermMatrix()`.

```
dtm <- DocumentTermMatrix(docs)
```

Some properties of the DTM are printed to learn more about it.

```
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 23, terms: 2984)>>
## Non-/sparse entries: 5363/63269
## Sparsity           : 92%
## Maximal term length: 56
## Weighting          : term frequency (tf)
## Sample             :
##                 Terms
## Docs             anim bitcoin can debian learn one packag system use will
##   asoiaf01.txt      0       0   3      0     0   3      0      0   0    0
##   asoiaf03.txt      0       0   1      0     0   2      0      0   1    5
##   bitcoin02.txt     0      52   6      0     0   6      0      0   6    6
##   churchill01.txt   0       0   1      0     0   0      0      0   0    3
##   covid01.txt       2       0   0      0     1   5      0      1   0    0
##   covid02.txt       0       0   0      0     0   0      0      0   0    1
##   covid03.txt      41       0   3      0     1  11      0      0   2    3
##   linux01.txt       0       0   4      0     0   5     16     14   2    3
##   linux02.txt       0       0  24    102     0  14     35     58  18    8
##   stats01.txt       0       0   1      0    11   2      1      0   3    2
```

```
freq <- colSums(as.matrix(dtm))
freq[head(order(freq), 20)]
```

```
##       aemon      appeas       arrog         ate         bed       belov
##           1           1           1           1           1           1
##       blame      bolton      bought      captiv    castamer      caster
##           1           1           1           1           1           1
##        chin       choke       clink      decept        defi discourtesi
##           1           1           1           1           1           1
##     distant         don
##           1           1
```

```
freq[tail(order(freq), 20)]
```

```
##     secur      make      time     build   softwar distribut    market    instal
##        38        39        39        39        39        41        42        42
##      also      anim     linux     learn       use      will   bitcoin    packag
##        43        45        45        47        48        48        52        52
##       one       can    system    debian
##        53        56        77       102
```

From `inspect(dtm)`, we learn that the DTM contains 2984 columns, indicating 2984 terms/tokens. It is very sparse, with a sparsity of 92%. `freq[head(order(freq), 20)]` and `freq[tail(order(freq), 20)]` shows the 20 least and most frequent terms respectively, in alphabetical order. The 20 least frequent terms appear

only once each, and their starting letters range from 'a' to 'd'. This is a strong indicator that there are many tokens appearing only once each. The most frequent term is `debian`, with 102 occurrences. However, this is believed to be a sparse term, as I am confident from my document collection process that only `linux02.txt` contains this token.

The next step is to remove sparse terms from the DTM. The maximal allowed sparsity, `sparse`, is set to 0.65, which gives a DTM with 54% sparsity and 32 tokens. A DTM closer to the guideline with 51% sparsity and 23 tokens can be obtained by setting `sparse` to 0.6, but through my analysis I observed that the 32-token DTM gives a better cluster dendrogram and network graph. **I also discovered that setting the sparsity even higher further improves the clustering, but I avoided this due to efficiency issues and poor readability of graphs.**

```
dtms <- removeSparseTerms(dtm, 0.65)
inspect(dtms)
```

```
## <<DocumentTermMatrix (documents: 23, terms: 32)>>
## Non-/sparse entries: 340/396
## Sparsity           : 54%
## Maximal term length: 5
## Weighting          : term frequency (tf)
## Sample             :
##                  Terms
## Docs              can just like make one peopl said time use will
##   asoiaf01.txt      3    0    1    2   3     1    3    0   0    0
##   asoiaf03.txt      1    3    3    1   2     0    3    1   1    5
##   bitcoin02.txt     6    0    8    4   6     3    1    2   6    6
##   churchill01.txt   1    1    1    0   0     1    0    3   0    3
##   covid01.txt       0    1    0    1   5     4   11    0   0    0
##   covid03.txt       3    3    4    3  11     3    1    3   2    3
##   linux01.txt       4    1    2    2   5     1    0    3   2    3
##   linux02.txt      24   14   12   13  14    15    3   19  18    8
##   peaky01.txt       0    0    0    1   1     0    0    0   0    4
##   queen01.txt       3    8    0    1   0     0    0    3   0    6
```

```
freqs <- colSums(as.matrix(dtms))
freqs[head(order(freqs), 20)]
```

```
##  mean  long   day world  year   see  well   now  sinc  take  need  mani  must
##    12    13    14    16    17    18    18    19    19    21    21    22    22
## think  know   old   two  come   may   new
##    22    23    23    23    24    25    26
```

```
freqs[tail(order(freqs), 20)]
```

```
##  must think  know   old   two  come   may   new  even   man peopl  said  just
##    22    22    23    23    23    24    25    26    27    29    30    32    33
##  like  make  time   use  will   one   can
##    34    39    39    48    48    53    56
```

Based on the output, the least frequent token is `mean` with 12 occurrences and the most frequent token is `can` with 56 occurrences. The full DTM after this step is included in the Appendix.

Task 4

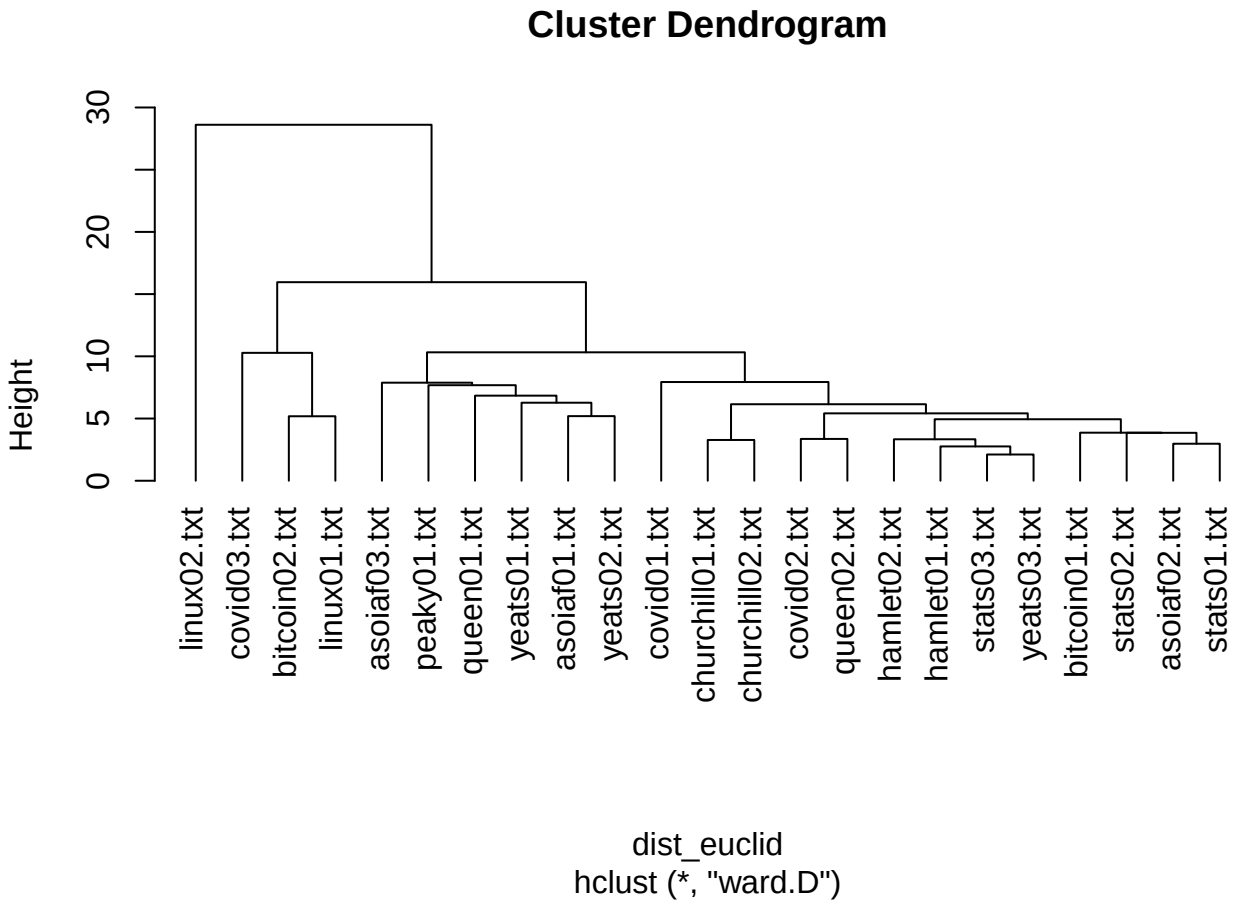
A hierarchical clustering of the corpus is done based on two metrics, Euclidean distance and cosine distance, to see which performs better. The DTM is first converted to a sstandard matrix format.

```
dtms_mat <- as.matrix(dtms)
```

In document clustering, each document is represented as a vector with many dimensions that correspond to terms it contains. Euclidean distance measures the straight-line distance between these vectors, grouping them based on closeness. A smaller distance represents higher similarity. Cosine distance measures the angle between vectors, with smaller angles corresponding to closer distance and higher similarity. Clustering with cosine distance can be further improved by weighting the DTM with the term frequency-inverse document frequency (TF-IDF) statistic beforehand, which assigns higher weights to terms that appear frequently within a document (implying importance), but rarely across all documents (implying significance).

The code for clustering using Euclidean distance is adapted from Lecture 10. `dtms_mat` is scaled and converted to a Euclidean distance matrix, and a dendrogram is plotted.

```
dist_euclid <- dist(scale(dtms_mat))
fit_euclid <- hclust(dist_euclid, method = "ward.D")
plot(fit_euclid, hang = -1)
```

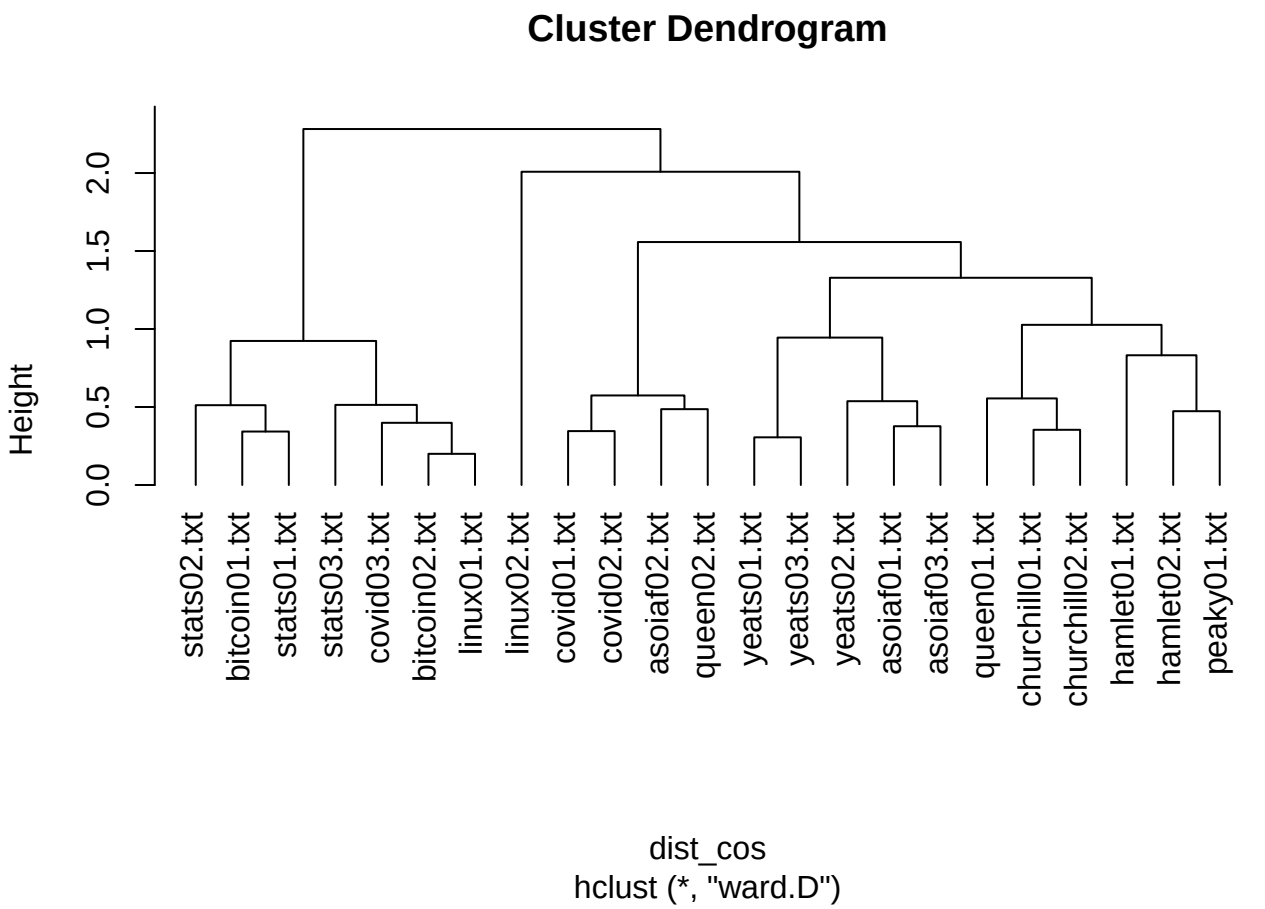


To cluster with TF-IDF weighting and cosine distance, the IDF for each term is first computed and a TF-IDF weighted matrix is obtained by applying the cross product of the TFs and IDFs. The formula for cosine distance is then applied to the matrix to get a cosine distance matrix. The dendrogram is then plotted. The code for this was adapted from this Stack Overflow thread. The code was studied and understood before application.

```
idf <- log(ncol(dtms_mat) / (1 + rowSums(dtms_mat != 0)))
idf <- diag(idf)
dtms_mat_tfidf <- crossprod(dtms_mat, idf)   # dtms_mat with weights
colnames(dtms_mat_tfidf) <- rownames(dtms_mat)
```

```
dist_cos <- 1 - crossprod(dtms_mat_tfidf) /
            (sqrt(colSums(dtms_mat_tfidf ** 2) %*%
            t(colSums(dtms_mat_tfidf ** 2))))
dist_cos[is.na(dist_cos)] <- 0
dist_cos <- as.dist(dist_cos)

fit_cos <- hclust(dist_cos, method = "ward.D")
plot(fit_cos, hang = -1)
```



At the highest level in Euclidean distance clustering, `linux02` is in a single cluster while all other documents are grouped into another cluster. This indicates that the clustering algorithm identifies `linux02` as an outlier document with distinct characteristics. However, it does share a common topic with `linux01`, so this is an inaccurate and imbalanced result. For the rest of the documents, some documents with the same topics are clustered together at low levels, such as `churchill01` and `churchill02`. However, there are some prominent inaccuracies, such as `bitcoin01` and `bitcoin02` being clustered far from each other (only grouped together at the highest level) and `asioaf02` (an extract from a fantasy novel) being clustered together with `stats01` (an extract from a statistics textbook) at the lowest level.

With cosine distance, the clustering produces more balanced clusters. `linux02` is grouped with several other documents at the highest level, but it is still separated from `linux01`. However, this clustering is overall better at grouping documents of the same topic together. For example, `bitcoin01` and `bitcoin02` are closer together, while the pairs of `covid01`-`covid02` and `yeats01`-`yeats03` are each in their own clusters at the lowest level. The overall height values of the clustering are also much smaller (with Euclidean distance, the largest height was 28.6 compared to 2.28 for cosine distance), indicating higher confidence in the grouping of similar documents. Conclusively, this is a better clustering for the documents compared to that with

Euclidean distance.

Since the actual topic of each document is known, it is possible to get a quantitative measure of each clustering by labelling each document with its topic, plotting a confusion matrix of the clustering, and computing the accuracy. Due to length, the confusion matrices and the cluster-topic assignments are shown in the Appendix (for Euclidean distance, 15 clusters are created to reduce topic-cluster ambiguity).

```
# function to remove suffix in document filename
short_name <- function(doc) {
    return(substr(doc, 1, nchar(doc) - 6))
}

doc_names <- list.files("docs")
doc_names_short <- unlist(lapply(doc_names, short_name))

table(Topic = doc_names_short, Cluster = cutree(fit_euclid, k = 15))
table(Topic = doc_names_short, Cluster = cutree(fit_cos, k = 10))
```

The accuracies of each matrix is calculated by hand. For clustering with Euclidean distance, this is given by

```
13 / 23
```

```
## [1] 0.5652174
```

Whereas for cosine distance, this is given by

```
15 / 23
```

```
## [1] 0.6521739
```

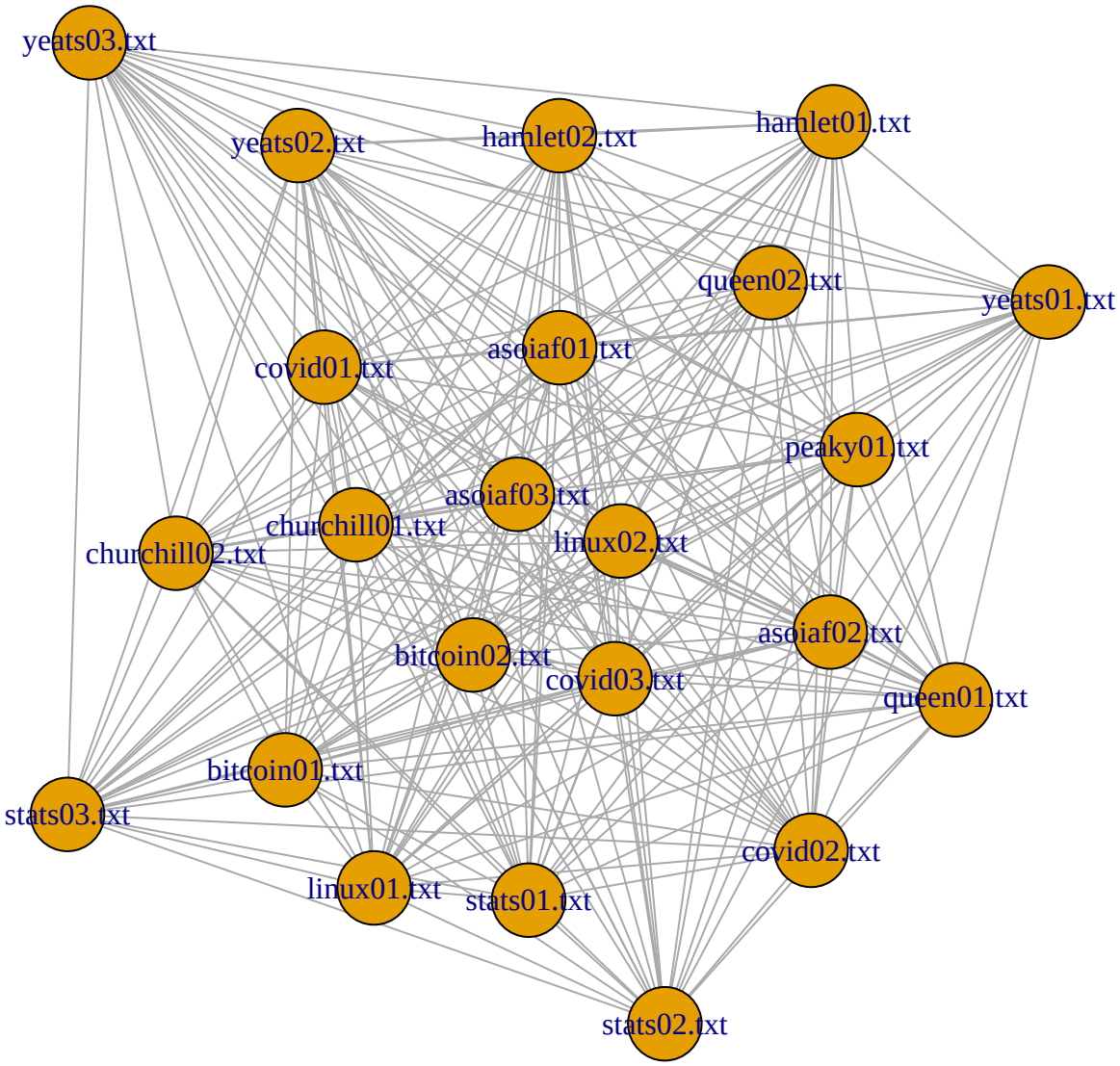
In agreement with the observations, clustering with cosine distance has a significantly higher accuracay than clustering with Euclidean distance.

Task 5

To create a single-mode network visualising the connections between documents based on the number of shared terms, the DTM is first converted to a binary matrix and then multiplied by its transpose. After the first step, a matrix which records 1 for each token that appears in that row's document is produced. After multiplication, the resulting matrix shows the number of shared tokens in each pair of documents. After getting this abstracts matrix, `graph_from_adjacency_matrix()` is used to plot the graph.

```
dtms_mat_bin <- as.matrix((dtms_mat > 0) + 0)
abs_mat <- dtms_mat_bin %*% t(dtms_mat_bin)
diag(abs_mat) <- 0

abs_net <- graph_from_adjacency_matrix(abs_mat, mode = "undirected",
                                       weighted = TRUE)
plot(abs_net)
```



The graph's high density can be immediately observed; almost every possible pair of vertices have an edge. Computing the density of the graph proves this.

```
graph.density(abs_net)
```

```
## [1] 0.9920949
```

This shows that, based on shared terms, each document is related to almost every other document to some extent. `linux02` and `asioaf03` are very close to each other compared to other vertex pairs, indicating a strong relationship, while `yeats03` seems to have relatively weaker connections to the other documents.

To identify clear groups in the data, a good start is to get the transitivity of the graph.

```
transitivity(abs_net)
```

```
## [1] 0.9923518
```

Transitivity, or clustering coefficient, is the proportion of triangles relative to the number of connected triples, a higher value of which indicates more tightly connected groups. The transitivity of this graph is very high at 0.992. The distribution of the documents into groups can be done in many ways, as in hierarchical clustering, but a more prominent group might be `yeats03`, `yeats02`, `hamlet02` and `hamlet01`, all of which are literary texts.

At first glance, `asioaf03` and `linux02` seem to be the most important documents due to their central positions. To get numerical measurements, the following code is run. Due to length, the outputs are shown in the Appendix.

```
sort(-closeness(abs_net))
sort(-betweenness(abs_net))
sort(evcent(abs_net)$vector)
sort(degree(abs_net))
```

CLoseness measures how well a document is connected to others, betweenness measures the document's potential to be an intermediary (and thus have more influence over the network flow), eigenvector centrality measures the quality of connections of a document, and degree is a vertex's number of connections. For closeness and betweenness, the results are made negative as the weights of the vertices correspond to the number of shared terms, meaning when two documents are closely connected, the distance (weight of the edge) between them is high.

For betweenness, 13 documents have values of 0, indicating that they all have maximum potential of being the network's "hubs". For degree, only `yeats03`, `bitcoin01` and `stats02` are not connected to every other document. However, this is not a strong indicator that they are less important, as their degrees are still very close to the maximum of 22.`yeats03` has the smallest value for all four metrics, so there is confidence that it is the least important document. On the contrary, `linux02` has the highest closeness and eigenvector centralities, a sign that it is possibly the most important document.

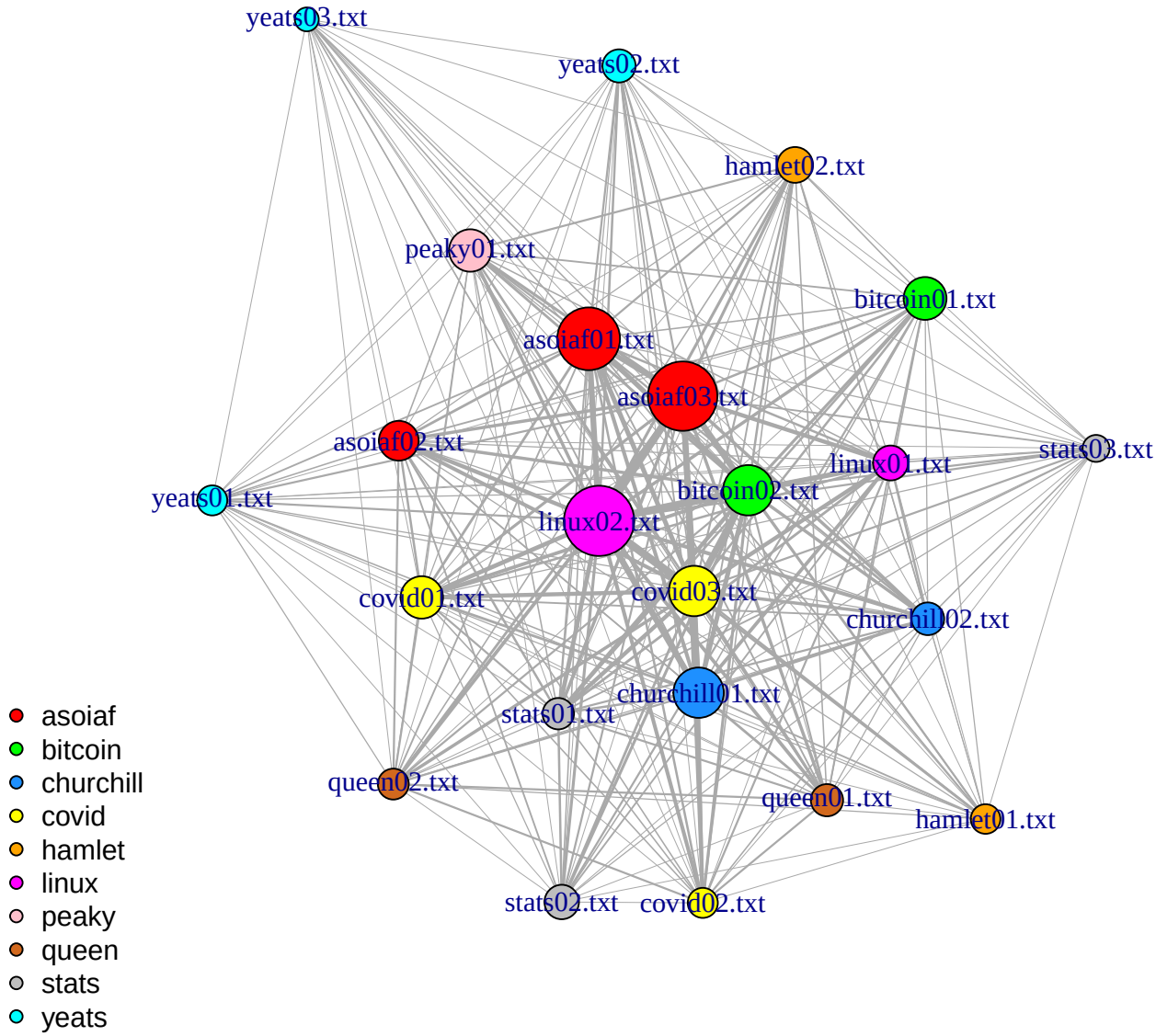
To visualise the interesting features of the data, the graph is improved by colouring the vertices based on topic, scaling the width of edges based on inter-document strength (number of shared terms) and scaling the vertex sizes based on the document's closeness centrality. The code to do this is given as follows.

```
doc_colr <- c("red", "green", "dodgerblue", "yellow", "orange", "magenta",
          "pink", "chocolate", "gray", "cyan")
topics <- unique(doc_names_short)

for (doc in doc_names) {
    V(abs_net)[doc]$color <- doc_colr[match(short_name(doc), topics)]
}

V(abs_net)$size <- 1 / closeness(abs_net, mode = "all") / 10
E(abs_net)$width <- E(abs_net)$weight / 6

plot(abs_net)
legend(x = -1.5, y = -0.5, legend = topics, pch = 21, cex = 1,
       pt.bg = doc_colr, bty = "n", ncol = 1)
```



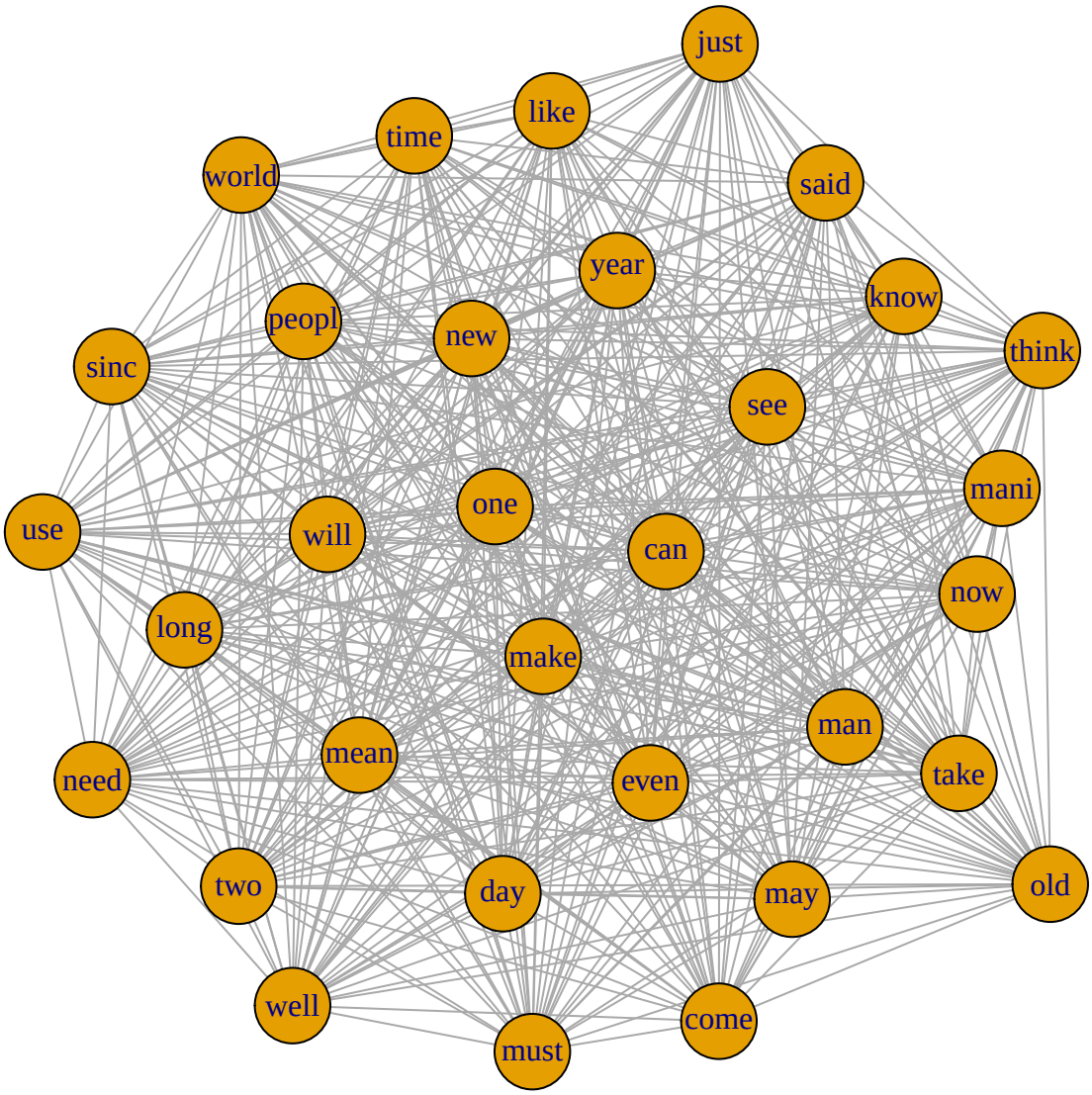
With this graph, we can more clearly understand the distance between the `yeats` documents and the others. It is also easier to observe that some topics have all documents close together (eg. `asioaf`) while others have the opposite (eg. `hamlet`).

Task 6

To create a single-mode network to illustrate the connections between the tokens based on the number of common documents they appear in, most of the code is reused from Task 5 with some adjustments. To create the tokens matrix, the transpose of the binary matrix is multiplied by the original. The graph is then plotted with the same function.

```
tok_mat <- t(dtms_mat_bin) %*% dtms_mat_bin
diag(tok_mat) <- 0

tok_net <- graph_from_adjacency_matrix(tok_mat, mode = "undirected",
                                       weighted = TRUE)
plot(tok_net)
```



This graph looks denser than the one for documents.

```
graph.density(tok_net)
```

```
## [1] 1
```

In line with the observation, the graph is indeed denser with a density of 1, meaning all vertices are interconnected. This tells us that the tokens are quite generic words, appearing across all documents of various topics.

Looking at the graph, it is difficult to identify clear groups as the vertices seem quite evenly spread out. A suspected cluster is `day`, `just` and `must` at the bottom of the graph, which forms a noticeable triangle.

```
transitivity(tok_net)
```

```
## [1] 1
```

The transitivity of the graph is 1. Even though it is hard to distinguish clusters by observation, this metric tells us that every three tokens can form a triangle, indicating a very high number of potential clusters. This is possibly also due to the genericness of the tokens. **I have experimented with manually removing some generic words during DTM pre-processing, but it lead to poorer clustering performance.**

The visually centre-most tokens are `can` and `peopl`, a sign that they may be relatively important. The following code is run to investigate this. The outputs are sohwn in the Appendix.

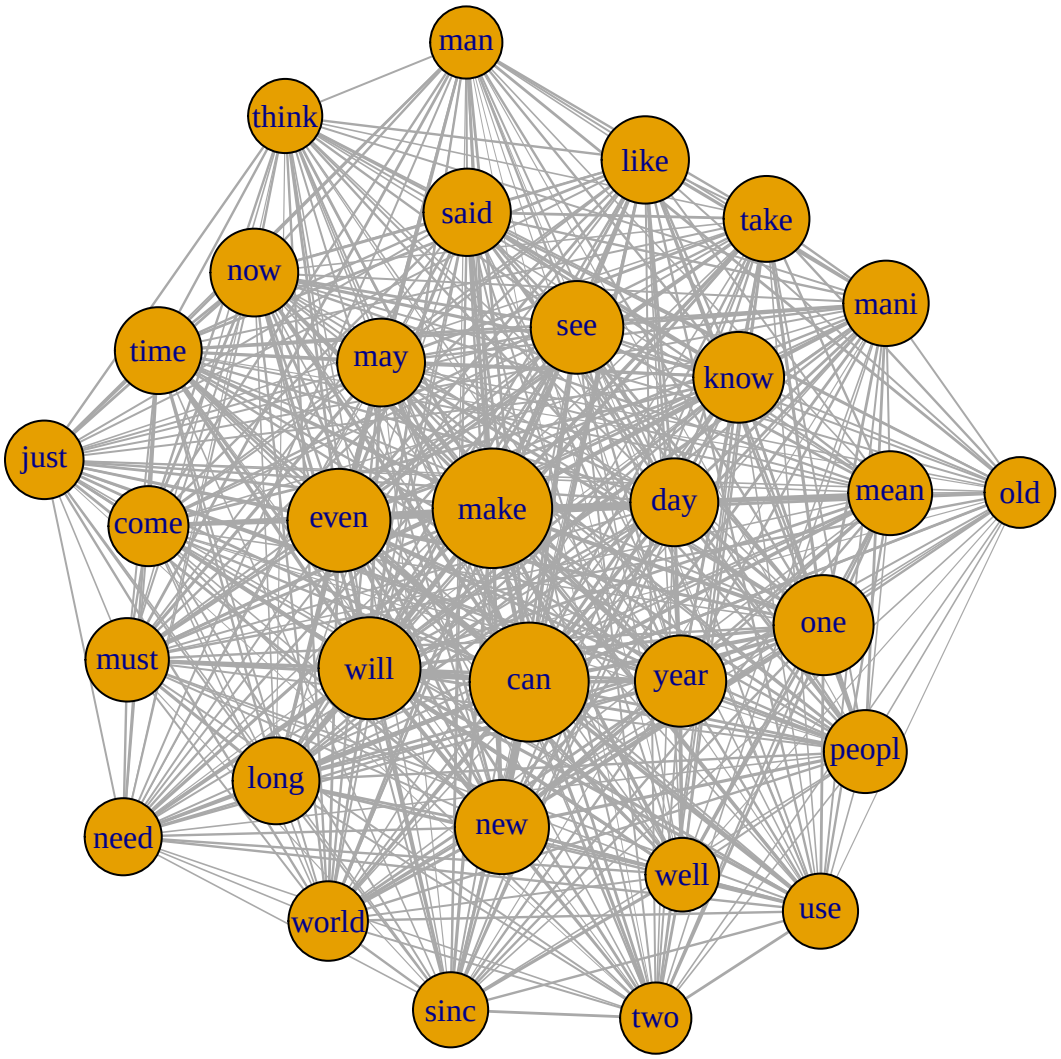
```
sort(-closeness(tok_net))
sort(-betweenness(tok_net))
sort(evcent(tok_net)$vector)
sort(degree(tok_net))
```

`can` has the second highest closeness centrality behind `make`, maximum betweenness centrality, the second highest eigenvector centrality also behind `make`, and the maximum degree. `peopl` performs worse than expected; relative to all other tokens, it has moderate closeness, betweenness and eigenvector centralities. Looking at the `make` vertex on hindsight, it is noticeable now that the edges surrounding this vertex are more layered and dense.

To improve this graph, the vertex sizes are scaled to their closeness centralities, and the edge widths are scaled to their weights (number of shared documents between tokens).

```
V(tok_net)$size <- 1 / closeness(tok_net, mode = "all") / 10
E(tok_net)$width <- E(tok_net)$weight / 5

plot(tok_net)
```



With this graph, it can now be clearly seen that `make` and `can` are two important central tokens, while `man` is notably further away from the rest and, as justified by the previously computed metrics, is relatively less important.

Task 7

To create a bipartite (two-mode) network graph, the data is first formatted using code adapted from Lecture 12.

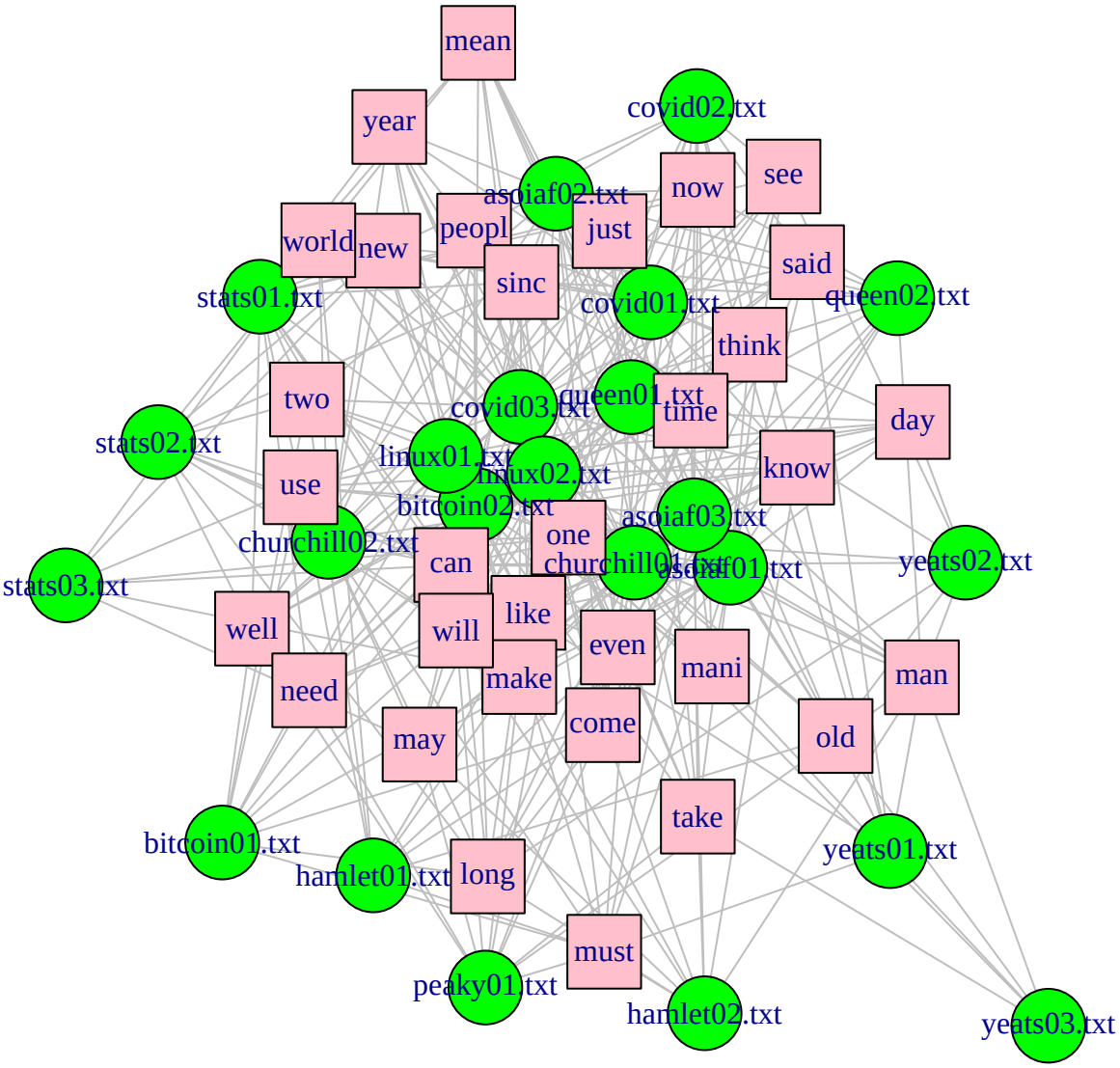
```
dtms_dfa <- as.data.frame(dtms_mat)
dtms_dfa$abstract <- rownames(dtms_dfa)
dtms_dfb <- data.frame()
for (i in seq_len(nrow(dtms_dfa))) {
    for (j in seq_len(ncol(dtms_dfa) - 1)) {
        to_use <- cbind(dtms_dfa[i, j], dtms_dfa[i, ncol(dtms_dfa)],
                        colnames(dtms_dfa[j]))
        dtms_dfb <- rbind(dtms_dfb, to_use)
    }
}
colnames(dtms_dfb) <- c("weight", "abstract", "token")

dtms_dfc <- dtms_dfb[dtms_dfb$weight != 0, ]
dtms_dfc <- dtms_dfc[, c("abstract", "token", "weight")]
```

In the above code, a data frame showing the frequency (weight) of each token in each document (`dtms_dfb`) is created. Rows with weights of 0 are removed in a new data frame (`dtms_dfc`) and this data frame is used to plot the bipartite graph.

```
bipart <- graph.data.frame(dtms_dfc, directed = FALSE)
V(bipart)$type <- bipartite_mapping(bipart)$type
V(bipart)$color <- ifelse(V(bipart)$type, "pink", "green")
V(bipart)$shape <- ifelse(V(bipart)$type, "square", "circle")
E(bipart)$color <- "grey"

plot(bipart)
```



As this graph is bipartite, the density and transitivity are expected to be very low. The closeness, betweenness and eigenvector centralities do not give us new insight, as the relationship between documents and that between tokens have already been analysed in Tasks 5 and 6. Hence, this graph is analysed by observation.

The overall structure of this graph resembles a "cluster" of documents in the middle surrounded by the tokens in a ring, all of which are surrounded by the remaining documents. The central cluster of documents are positioned as such as they are linked to most of the tokens. This cluster includes `linux02`, `asioaf03` and `covid03`, which the metrics computed in Task 5 determined to be the relatively more important documents. Similar to the graph in Task 5, `yeats03` lies relatively further away from the other vertices, with links to fewer tokens. The degrees of the vertices prove this (output shown in Appendix).

```
sort(degree(bipart))
```

`linux02` and `covid03` have the two highest degrees while `yeats03` has the least.

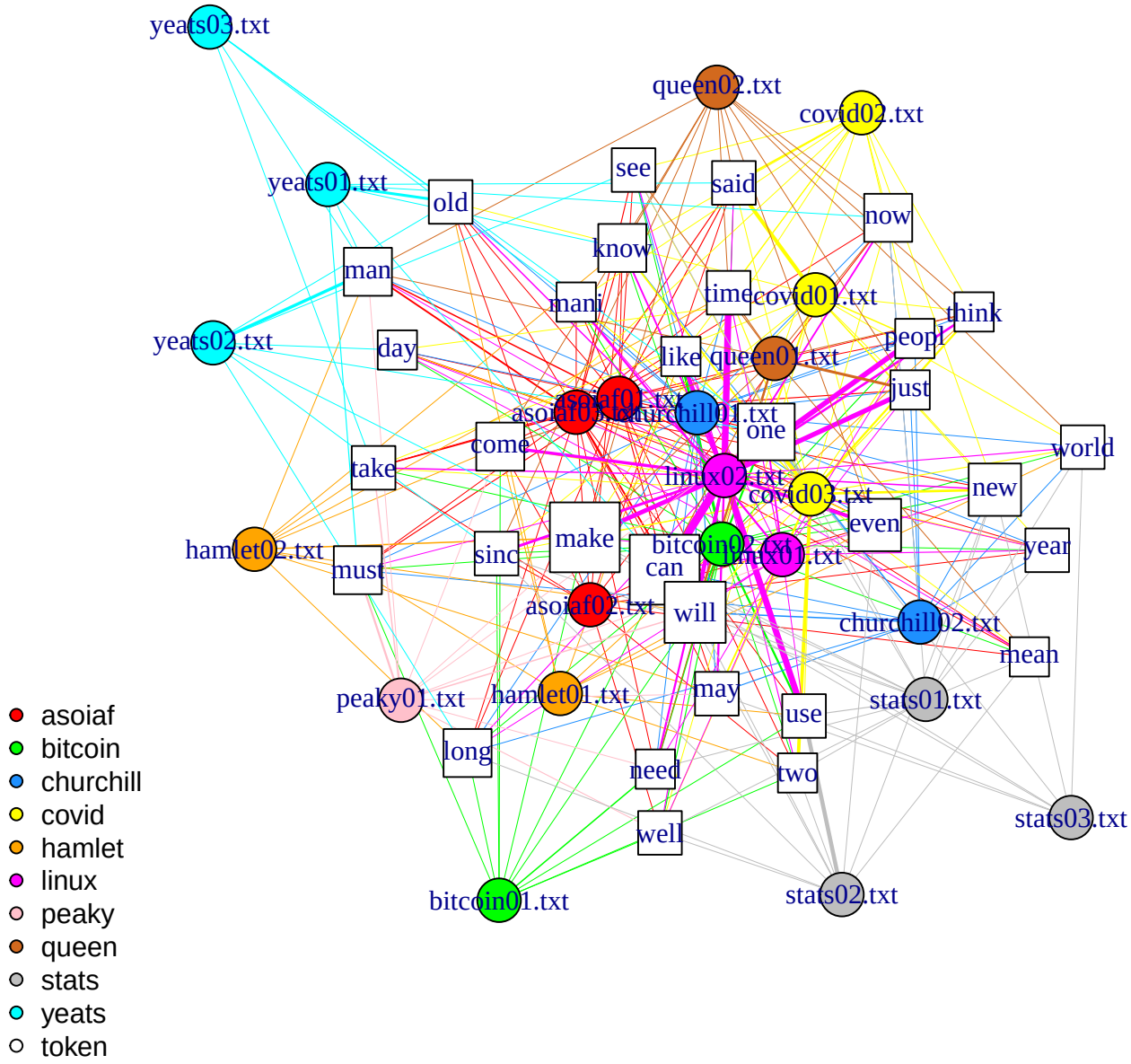
Other notable groups of documents which are not within the central cluster include the group of all three `stats` documents, which are situated close together on the left of the graph, and the pair of `asoiaf02` and `covid02` at the top, which are separated from the other documents by a "wall" of tokens.

To improve this graph, document vertices are made smaller for readability and coloured as in Task 5, token vertices are scaled according to their degree and re-coloured white, edges are coloured based on the document

vertex it connects to, and edge widths are scaled according to the weight of the edge (frequency of token in document). Document vertex sizes are not scaled as the observable amount of coloured outgoing edges gives a good idea of their degrees.

```
for (i in seq_len(23)) V(bipart)$size[i] <- 10
V(bipart)$size[24:55] <- degree(bipart)[24:55]
for (j in seq_len(length(doc_names))) {
    V(bipart)[j]$color <- doc_colr[match(short_name(doc_names[j]), topics)]
}
for (k in c(24:length(V(bipart)))) V(bipart)[k]$color <- "white"
E(bipart)$width <- as.numeric(dtms_dfc$weight) / 5
E(bipart)$color <- tail_of(bipart, E(bipart))$color

plot(bipart)
legend(x = -1.5, y = -0.5, legend = c(topics, "token"), pch = 21, cex = 1,
       pt.bg = c(doc_colr, "white"), bty = "n", ncol = 1)
```



From this improved graph, the high relative importance of `linux02` is more clearly seen, having edges that are significantly thicker than its close neighbour `bitcoin02`, which is considered to be of high importance as well. `make`, `can` and `will` forms a triple of tokens that appear the most across all documents. Some documents, such as `churchill`, are within the central cluster despite having thin edges, indicating that they contain smaller numbers of many tokens. This contrasts with outer documents such as the `yeats` documents, which have thick edges but not that many edges in total, indicating a larger numbers of fewer tokens.

Task 8

Hierarchical clustering can be said to be the quicker and easier way to get an understanding of the relationships between documents and tokens in the corpus. It neatly groups documents into clusters at different heights, so groupings can be interpreted based on the desired cluster size. The accuracies of the clusterings performed in Task 4 are not very high, but an accuracy of around 0.65 for clustering with cosine distance can be considered strong for a small corpus (only 23 documents) and the genericness of the tokens.

However, clustering does not identify important documents, tokens or groups. Social network analysis visualises relationships between documents, tokens and document-token pairs, allowing viewers to quickly gain insight about the relationships within the corpus. If needed, viewers can access computable metrics (eg. closeness, betweenness, transitivity, etc.) to gauge the overall connectedness of the network or the importance of each document/token. This makes social networks a flexible way to identify important groups and relationships in the data that can be utilised by more viewers, be they a general audience or a technical one.

Appendix

Reference list of documents collected in Task 1.

- ArchWiki. (2023, May 16). *Arch Linux.* ArchWiki. https://wiki.archlinux.org/title/Arch_Linux
- Frayer, J. M., & Jett, J. (2023, May 26). *China faces a new Covid wave that could peak at 65 million cases a week.* NBC News. https://www.nbcnews.com/news/world/china-covid-second-wave-xbb-variant-omicron-rcna86171
- Gimpel, J. (2022, July 26). *Statistical Learning Theory - The basis for neural networks.* Medium; Towards Data Science. https://towardsdatascience.com/statistical-learning-theory-26753bdee66e
- Hooson, M. (2023, April 3). *What Is Bitcoin (BTC) And How Does It Work?* Forbes Advisor Australia. https://www.forbes.com/advisor/au/investing/cryptocurrency/what-is-bitcoin/
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An Introduction to Statistical Learning with Applications in R* (2nd ed., pp. vii–viii). Springer.
- Knight, S. (Writer), Byrne, A. (Director). (2022, March 6). Black Shirt (Season 6, Episode 2) [TV series episode]. In C. Mandabach, G. Brennan, S. Knight, C. Murphy, J. Glazebrook, F. Tiplady (Executive Producers), *Peaky Blinders.* Caryn Mandabach Productions; Tiger Aspect Productions; Screen Yorkshire.
- Lewis, T. (2023, April 12). *What New Evidence from the Wuhan Market Tells Us about COVID's Origins.* Scientific American. https://www.scientificamerican.com/article/what-new-evidence-from-the-wuhan-market-tells-us-about-covids-origins1/
- Mai, J., & Lew, L. (2020, January 4). *Chinese city at centre of pneumonia outbreak remains calm.* South China Morning Post. https://www.scmp.com/news/china/science/article/3044703/chinese-city-centre-mysterious-pneumonia-outbreak-remains-calm
- Martin, G. R. R. (1996). Catelyn XI. In *A Game of Thrones* (pp. 656–666). HarperCollins.
- Martin, G. R. R. (2005). Brienne V. In *A Feast for Crows* (pp. 363–376). HarperCollins.
- Martin, G. R. R. (2011). Davos IV. In *A Dance with Dragons* (pp. 382–394). HarperCollins.
- Nakamoto, S. (2008). Bitcoin: a Peer-to-Peer Electronic Cash System. In *bitcoin.org.* https://bitcoin.org/bitcoin.pdf
- Queen. (1975). Bohemian Rhapsody [Song]. On *A Night at the Opera.* EMI.
- Queen. (1978). Spread Your Wings [Song]. On *News of the World.* EMI.
- Shakespeare, W. (n.d.). *To be, or not to be, that is the question* [Speech]. In *Hamlet* (Act III, Scene i).

Poetry Foundation. https://www.poetryfoundation.org/poems/56965/speech-to-be-or-not-to-be-that-is-the-question
- Shakespeare, W. (2022). *Hamlet's First Soliloquy* [Speech]. In *Hamlet* (Act I, Scene ii). Owlcation. https://owlcation.com/humanities/Hamlets-1st-Soliloquy
- Srivastava, M. (2022, July 3). *WhyDebian.* Debian Wiki. https://wiki.debian.org/WhyDebian
- *What is statistical learning? Definition and examples.* (n.d.). Market Business News. Retrieved May 29, 2023, from https://marketbusinessnews.com/financial-glossary/statistical-learning/
- Winston, C. (1940, June 18). *This was their finest hour* [Speech]. https://en.wikipedia.org/wiki/This_was_their_finest_hour#Peroration
- Winston, C. (1940, June 4). *We shall fight on the beaches* [Speech]. https://en.wikipedia.org/wiki/We_shall_fight_on_the_beaches#Peroration
- Yeats, W. B. (1916). *The Fisherman.* Poetry Foundation. https://www.poetryfoundation.org/poetrymagazine/poems/13324/the-fisherman
- Yeats, W. B. (1989). *The Circus Animals' Desertion.* Poetry Foundation. https://www.poetryfoundation.org/poems/43299/the-circus-animals-desertion
- Yeats, W. B. (1989). *When You Are Old.* Poetry Foundation. https://www.poetryfoundation.org/poems/43283/when-you-are-old

Document-term matrix at the end of Task 3, printed as a data frame.

```
as.data.frame(as.matrix(dtms))
```

```
##                 can day even know like long make man mani may mean must old one
## asoiaf01.txt      3   1    3    1    1    1    2   4    1   1    2    3   1   3
## asoiaf02.txt      1   0    1    1    0    0    1   0    0   0    1    0   1   0
## asoiaf03.txt      1   5    1    3    3    1    1   5    2   1    1    1   1   2
## bitcoin01.txt     1   0    0    0    0    1    2   0    0   0    0    1   0   0
## bitcoin02.txt     6   1    1    1    8    1    4   0    3   2    1    1   0   6
## churchill01.txt   1   1    2    0    1    1    0   1    3   3    0    1   2   0
## churchill02.txt   1   0    0    1    0    1    0   0    0   3    0    1   0   1
## covid01.txt       0   1    2    0    0    0    1   0    0   0    0    0   2   5
## covid02.txt       0   0    0    1    1    0    1   0    1   0    0    0   0   0
## covid03.txt       3   1    3    5    4    0    3   0    3   5    2    0   0  11
## hamlet01.txt      0   0    1    0    2    0    0   0    0   0    0    2   1   0
## hamlet02.txt      0   0    0    1    0    1    4   1    0   1    0    1   0   0
## linux01.txt       4   1    0    0    2    1    2   0    6   0    0    0   0   5
## linux02.txt      24   2   10    5   12    3   13   1    2   6    2    2   4  14
## peaky01.txt       0   0    1    0    0    0    1   1    0   2    0    6   0   1
## queen01.txt       3   0    0    0    0    0    1   2    0   0    1    0   0   0
## queen02.txt       0   0    1    3    0    0    1   2    0   0    0    0   0   0
## stats01.txt       1   0    0    0    0    0    1   0    0   0    1    0   0   2
## stats02.txt       2   0    1    0    0    1    0   0    0   0    1    0   0   0
## stats03.txt       2   0    0    0    0    0    1   0    0   1    0    0   0   1
## yeats01.txt       2   0    0    1    0    0    0   2    0   0    0    3   8   0
## yeats02.txt       1   1    0    0    0    1    0   9    0   0    0    0   1   1
## yeats03.txt       0   0    0    0    0    0    0   1    1   0    0    0   2   1
##                 peopl said see take think two come now sinc well year just need
## asoiaf01.txt        1    3   2    1     2   1    0   0    0    0    0    0    0
## asoiaf02.txt        0    2   0    1     0   2    1   1    1    1    1    0    0
## asoiaf03.txt        0    3   3    4     0   0    2   1    0    0    1    3    1
## bitcoin01.txt       0    0   0    0     0   1    1   0    1    1    0    0    4
## bitcoin02.txt       3    1   1    2     0   2    0   0    1    3    2    0    4
## churchill01.txt     1    0   1    0     1   0    0   1    0    0    1    1    1
## churchill02.txt     1    0   0    0     0   0    0   1    0    0    1    1    0
## covid01.txt         4   11   0    1     1   0    0   0    2    1    2    1    0
```

```
## covid02.txt         1    5   2    0     1   0    1   2    0    0    0    0    0
## covid03.txt         3    1   1    1     6  10    2   2    1    2    3    3    0
## hamlet01.txt        0    0   0    0     1   2    2   0    0    0    0    0    0
## hamlet02.txt        0    0   0    2     0   0    1   0    0    0    0    0    0
## linux01.txt         1    0   0    0     0   0    0   0    0    2    3    1    1
## linux02.txt        15    3   3    4     7   3    9   5    8    3    2   14    6
## peaky01.txt         0    0   0    4     0   0    1   0    0    3    0    0    1
## queen01.txt         0    0   3    0     2   0    3   2    0    0    0    8    1
## queen02.txt         0    2   0    0     1   0    1   2    1    0    0    1    0
## stats01.txt         0    0   1    0     0   1    0   1    1    1    1    0    2
## stats02.txt         0    0   0    0     0   1    0   0    1    1    0    0    0
## stats03.txt         0    0   0    0     0   0    0   0    0    0    0    0    0
## yeats01.txt         0    1   0    0     0   0    0   1    0    0    0    0    0
## yeats02.txt         0    0   1    0     0   0    0   0    2    0    0    0    0
## yeats03.txt         0    0   0    1     0   0    0   0    0    0    0    0    0
##                 new time use will world
## asoiaf01.txt      0    0   0    0     0
## asoiaf02.txt      0    0   0    0     0
## asoiaf03.txt      1    1   1    5     0
## bitcoin01.txt     0    0   2    1     0
## bitcoin02.txt     3    2   6    6     1
## churchill01.txt   1    3   0    3     1
## churchill02.txt   1    0   0    4     2
## covid01.txt       1    0   0    0     2
## covid02.txt       2    2   0    1     0
## covid03.txt       7    3   2    3     3
## hamlet01.txt      0    0   1    0     1
## hamlet02.txt      0    1   0    1     0
## linux01.txt       2    3   2    3     0
## linux02.txt       4   19  18    8     3
## peaky01.txt       0    0   0    4     0
## queen01.txt       0    3   0    6     0
## queen02.txt       0    2   0    0     1
## stats01.txt       2    0   3    2     1
## stats02.txt       1    0  12    1     0
## stats03.txt       1    0   1    0     1
## yeats01.txt       0    0   0    0     0
## yeats02.txt       0    0   0    0     0
## yeats03.txt       0    0   0    0     0
```

Confusion matrices used for quantitative measure of clustering in Task 4.

```
table(Topic = doc_names_short, Cluster = cutree(fit_euclid, k = 10))
```

```
##            Cluster
## Topic       1 2 3 4 5 6 7 8 9 10
##   asoiaf    1 1 1 0 0 0 0 0 0  0
##   bitcoin   0 1 0 1 0 0 0 0 0  0
##   churchill 0 2 0 0 0 0 0 0 0  0
##   covid     0 1 0 0 1 1 0 0 0  0
##   hamlet    0 2 0 0 0 0 0 0 0  0
##   linux     0 0 0 1 0 0 1 0 0  0
##   peaky     0 0 0 0 0 0 0 1 0  0
##   queen     0 1 0 0 0 0 0 0 1  0
##   stats     0 3 0 0 0 0 0 0 0  0
```

```
##   yeats     1 1 0 0 0 0 0 0 0  1
```

```
table(Topic = doc_names_short, Cluster = cutree(fit_cos, k = 10))
```

```
##            Cluster
## Topic       1 2 3 4 5 6 7 8 9 10
##   asoiaf    2 1 0 0 0 0 0 0 0  0
##   bitcoin   0 0 1 1 0 0 0 0 0  0
##   churchill 0 0 0 0 2 0 0 0 0  0
##   covid     0 0 0 1 0 2 0 0 0  0
##   hamlet    0 0 0 0 0 0 1 1 0  0
##   linux     0 0 0 1 0 0 0 0 1  0
##   peaky     0 0 0 0 0 0 0 1 0  0
##   queen     0 1 0 0 1 0 0 0 0  0
##   stats     0 0 2 1 0 0 0 0 0  0
##   yeats     1 0 0 0 0 0 0 0 0  2
```

Euclidean distance cluster assignment for each topic, from Task 4.

- **1** asoiaf
- **2** stats
- **3** NA
- **4** bitcoin
- **5** churchill
- **6** covid
- **7** NA
- **8** NA
- **9** hamlet
- **10** linux
- **11** NA
- **12** peaky
- **13** queen
- **14** yeats
- **15** NA

Cosine distance cluster assignment for each topic, from Task 4.

- **1** asioaf
- **2** queen
- **3** stats
- **4** bitcoin
- **5** churchill
- **6** covid
- **7** hamlet
- **8** peaky
- **9** linux
- **10** yeats

Outputs for closeness, betweeness, eigenvector centralities and degree measurements of the abstracts network for Task 5.

```
sort(-closeness(abs_net))
```

```
##     yeats03.txt     stats03.txt    hamlet01.txt     covid02.txt     yeats01.txt
##    -0.018518519    -0.016393443    -0.014925373    -0.014705882    -0.014705882
##     queen02.txt     stats01.txt     queen01.txt churchill02.txt     yeats02.txt
##    -0.014285714    -0.014084507    -0.013698630    -0.013513514    -0.013333333
##     stats02.txt     linux01.txt    hamlet02.txt    asoiaf02.txt     covid01.txt
```

```
##    -0.012820513    -0.012658228    -0.012345679    -0.011111111    -0.010416667
##     peaky01.txt   bitcoin01.txt   bitcoin02.txt churchill01.txt     covid03.txt
##    -0.010416667    -0.010309278    -0.008771930    -0.008771930    -0.008771930
##    asoiaf01.txt    asoiaf03.txt     linux02.txt
##    -0.007042254    -0.006369427    -0.006289308
```

```
sort(-betweenness(abs_net))
```

```
##     yeats03.txt     yeats01.txt     stats03.txt    hamlet01.txt     covid02.txt
##    -124.3388889     -32.5563492     -24.2468254      -5.8333333      -5.2111111
##     yeats02.txt     queen02.txt     stats02.txt    hamlet02.txt     covid01.txt
##      -4.0095238      -1.1277778      -1.0928571      -1.0833333      -0.1428571
##    asoiaf01.txt    asoiaf02.txt    asoiaf03.txt   bitcoin01.txt   bitcoin02.txt
##       0.0000000       0.0000000       0.0000000       0.0000000       0.0000000
## churchill01.txt churchill02.txt     covid03.txt     linux01.txt     linux02.txt
##       0.0000000       0.0000000       0.0000000       0.0000000       0.0000000
##     peaky01.txt     queen01.txt     stats01.txt
##       0.0000000       0.0000000       0.0000000
```

```
sort(evcent(abs_net)$vector)
```

```
##     yeats03.txt     yeats01.txt     stats03.txt     yeats02.txt    hamlet01.txt
##       0.1691727       0.2494883       0.2822704       0.2858423       0.2920158
##    hamlet02.txt     stats02.txt   bitcoin01.txt     peaky01.txt     queen02.txt
##       0.3638805       0.3667513       0.3973789       0.3988274       0.4099634
##     queen01.txt     covid02.txt churchill02.txt    asoiaf02.txt     covid01.txt
##       0.4293494       0.4638567       0.4742254       0.4885642       0.5109625
##     stats01.txt     linux01.txt    asoiaf01.txt churchill01.txt    asoiaf03.txt
##       0.5372105       0.5729446       0.6682947       0.6938792       0.8580945
##   bitcoin02.txt     covid03.txt     linux02.txt
##       0.8593957       0.8803064       1.0000000
```

```
sort(degree(abs_net))
```

```
##     yeats03.txt   bitcoin01.txt     stats02.txt    asoiaf01.txt    asoiaf02.txt
##              20              21              21              22              22
##    asoiaf03.txt   bitcoin02.txt churchill01.txt churchill02.txt     covid01.txt
##              22              22              22              22              22
##     covid02.txt     covid03.txt    hamlet01.txt    hamlet02.txt     linux01.txt
##              22              22              22              22              22
##     linux02.txt     peaky01.txt     queen01.txt     queen02.txt     stats01.txt
##              22              22              22              22              22
##     stats03.txt     yeats01.txt     yeats02.txt
##              22              22              22
```

Outputs for closeness, betweeness, eigenvector centralities and degree measurements of the tokens matrix for Task 6.

```
sort(-closeness(tok_net))
```

```
##          old          two          man         well        think         sinc
## -0.006896552 -0.006849315 -0.006756757 -0.006622517 -0.006578947 -0.006493506
##          use         need         just        world         come        peopl
## -0.006493506 -0.006329114 -0.006211180 -0.006134969 -0.006097561 -0.005882353
##         must         mean         mani         take         long         time
## -0.005847953 -0.005813953 -0.005714286 -0.005681818 -0.005617978 -0.005617978
##         like         said          day          may          now         know
```

```
## -0.005586592 -0.005586592 -0.005555556 -0.005555556 -0.005555556 -0.005376344
##         year          see          new          one         will         even
## -0.005347594 -0.005263158 -0.005181347 -0.004878049 -0.004784689 -0.004739336
##          can         make
## -0.004098361 -0.004081633
```

```
sort(-betweenness(tok_net))
```

```
##        man        old      think        two       come       sinc       need
## -24.250000 -18.666667 -12.250000  -8.166667  -6.416667  -5.750000  -3.750000
##       well        use        day      peopl       just        can       even
##  -2.916667  -2.166667  -0.500000  -0.500000  -0.500000   0.000000   0.000000
##       know       like       long       make       mani        may       mean
##   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
##       must        one       said        see       take        now       year
##   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
##        new       time       will      world
##   0.000000   0.000000   0.000000   0.000000
```

```
sort(evcent(tok_net)$vector)
```

```
##       old     think       man       two      sinc      need      just      come
## 0.5925947 0.6128913 0.6206450 0.6305491 0.6513832 0.6595361 0.6707976 0.6727737
##     world      well       use      mean     peopl      must      take      mani
## 0.6776785 0.6830894 0.6914317 0.7037976 0.7071126 0.7076312 0.7114846 0.7170149
##      like       now      said       day      long       may      time      know
## 0.7307381 0.7331570 0.7334931 0.7354692 0.7402329 0.7442322 0.7495378 0.7636665
##       see      year       one      even       new      will       can      make
## 0.7678671 0.7898673 0.8405716 0.8408895 0.8473481 0.9219252 0.9934387 1.0000000
```

```
sort(degree(tok_net))
```

```
##   can   day  even  know  like  long  make   man  mani   may  mean  must   old
##    31    31    31    31    31    31    31    31    31    31    31    31    31
##   one peopl  said   see  take think   two  come   now  sinc  well  year  just
##    31    31    31    31    31    31    31    31    31    31    31    31    31
##  need   new  time   use  will world
##    31    31    31    31    31    31
```

Degrees of vertices of the bipartite graph in Task 7.

```
sort(degree(bipart))
```

```
##     yeats03.txt     stats03.txt     yeats01.txt     yeats02.txt    hamlet01.txt
##               5               7               7               8               9
##             day            like            mani            mean           peopl
##               9               9               9               9               9
##           think             two            just            need    hamlet02.txt
##               9               9               9               9              10
##     stats02.txt             may             old            said             see
##              10              10              10              10              10
##            take            sinc            well            year            time
##              10              10              10              10              10
##             use           world   bitcoin01.txt     peaky01.txt            know
##              10              10              11              11              11
##            long             man            must            come             now
##              11              11              11              11              11
```

```
##     queen01.txt     queen02.txt            even             new churchill02.txt
##              12              12              12              12              13
##     covid02.txt             one    asoiaf02.txt            will     covid01.txt
##              13              13              14              14              15
##     stats01.txt     linux01.txt             can            make    asoiaf01.txt
##              15              16              16              16              20
## churchill01.txt    asoiaf03.txt   bitcoin02.txt     covid03.txt     linux02.txt
##              21              26              26              27              32
```