Network Analysis of the CRAN Ecosystem and Task View Classification

Dylan Dijk

Student ID: 1802183

Email: Dylan.Dijk@warwick.ac.uk

Supervised by Professor Ioannis Kosmidis

Report submitted in partial fulfillment of the requirements for the degree of MMORSE at the University of Warwick

Department of Statistics

University of Warwick

Coventry, United Kingdom

May 2022

Abstract

The number of packages hosted on the Comprehensive R Archive Network (CRAN) has grown very large since it was created. To help organise these packages CRAN introduced Task Views which group together packages used for similar purposes. The infrastructure of the development and maintenance of the CRAN Task Views moved to GitHub towards the end of 2021. This move to GitHub made it possible to follow discussions between the Task View maintainers and other members of the R community. This uncovered some of the challenges that are encountered in the maintenance of the CRAN Task Views. Task View editors need to make decisions on which Task Views to include whilst trying to avoid any two Task Views being too similar. In this report we create a similarity measure between Task Views and compare the results to decisions made by the Task View editors. In addition, Task View maintainers need to select packages to add to their Task View. In this report we use multinomial regression to build a multi-class classifier of packages to Task Views. This can be used to provide recommendations for packages to Task Views. On a hold-out test set the accuracy of the model is 80%.

Acknowledgements

I would like to say thank you to Professor Ioannis Kosmidis for his guidance, support and feedback throughout the project.

Contents

1 Introduction

1.1 The Comprehensive R Archive Network

The Comprehensive R Archive Network (CRAN) is the central software repository for the programming language R (R Core Team, 2020). CRAN contains up-to-date versions of code and documentation for R. The content is stored in a network of servers around the world, known as the CRAN Mirrors (see the CRAN homepage[1], Section "What are R and CRAN"), with each server containing identical information.

In addition, CRAN contains a package repository which stores packages that can be installed to extend the functionalities of base R[2]. R packages are also hosted in other repositories, such as: Bioconductor[3], Omegahat[4], R-Forge[5] and GitHub[6]. However, the CRAN package repository is the main repository for R, with the packages stored there being easily installed by users and tested daily on multiple systems.

The number of packages in CRAN has been growing exponentially over the last 10 years. Currently, there are more than 18800 packages (an up to date figure is given at the top of the CRAN package web page[7]) with contributions from over 27,000 authors.

In the rest of this introductory section we describe the data that is available surrounding the packages hosted on CRAN. We then cover some of the tools we can use to extract and analyse this data. We also introduce Task Views, which are groups of packages used for similar purposes, and highlight some of the challenges that are faced by the Task View editors and maintainers.

In Section 2 we present the results of an exploratory analysis of CRAN from two snapshots which provides an overview of the data. In this section we also present some results of a social network analysis using the CRAN collaborative network.

One of the challenges faced by CRAN Task View editors is to decide which Task Views to include whilst making sure no two Task Views are too similar. Another common issue is

[1]CRAN mirrors: `https://cran.r-project.org/` - section "What are R and CRAN"
[2]List of Base R packages: `https://cran.r-project.org/doc/FAQ/R-FAQ.html#Add_002don-packages-in-R`
[3]Bioconductor repository: `https://master.bioconductor.org/install/`
[4]Omegahat repository: `http://www.omegahat.net/`
[5]R-Forge repository: `https://r-forge.r-project.org/`
[6]GitHub: `https://github.com/`
[7]Current number of packages available in CRAN given at top of web page: `https://cran.r-project.org/web/packages/`

Task Views not being properly maintained which then leads to them not having a sharp focus.

In Section 3 we define a similarity measure for Task Views utilising the package dependencies and the text data from the Task View web pages. After creating this similarity measure, we compare the results to decisions made by the Task View maintainers.

The Task View maintainers need to decide which packages to include in their Task View. They need to keep their Task View up to date as packages are archived and new packages are released (see issue 5[8] by the Environmetrics Task View[9] maintainer). Due to the huge number of packages that are in CRAN it is infeasible for a Task View maintainer to review all of the available packages. Therefore, a model that would give a small selection of high quality suggestions for the maintainer to then review would be useful.

In Section 4 we build a multi-class classifier using multinomial regression to classify R packages to Task Views. We construct features using the data described in the previous sections.

[8]Comment by the Environmetrics Task View maintainer in regards to keeping up to date with packages: https://github.com/cran-task-views/Environmetrics/issues/5

[9]Environmetrics Task View web page: https://cran.r-project.org/web/views/Environmetrics.html

1.2 Package Directive and Author Collaboration Networks

When developing a package, authors need to document precisely how their package utilises or enhances existing packages. They must create a description file listing the packages that theirs communicates with. Listing 1 on the next page shows part of the description file for the **lubridate** R package (Grolemund and Wickham, 2011) as an example.

There are five possible types of relation between packages that can be given in the description file, and they are listed below (see, Writing R Extensions, Section 1.1.3[10] for a more detailed description on the different package dependencies):

1. **Depends** - Packages listed in this field will be loaded and attached before the current package, therefore users will be able to access functions from these packages. This field is also used to indicate dependency on a particular version of R, for example Listing 1 shows **lubridate** depends on version 3.2 of R or higher.

2. **Imports** - These are packages that are loaded into memory but are not attached. In addition to the description file a namespace file is then required to make sure the correct functions are used by the functions within the package.

3. **Linking to** - Packages where the current package can use their code. For example the package can use the C++ code written in someone else's package.

4. **Suggests** - The package can be used with these packages but does not require them. These packages might be needed for example to run tests and vignettes.

5. **Enhances** - Packages that can be extended by the current package. For example by providing methods for classes from these packages.

The first three types of package dependencies are referred to as **Hard Dependencies**, because any package listed in any of these fields need to be installed alongside the current package. The last two types are called **Soft Dependencies**, as they are not required to use the package.

Listing 1 shows a part of the description file for the **lubridate** R package (Grolemund and Wickham, 2011). In this example there is a package listed in all five of the dependency fields. Listing 1 shows just a selection of lines taken from the complete description file which includes additional information such as the list of authors. The full description file

[10]Package dependency documentation: https://cran.r-project.org/doc/manuals/r-release/R-exts.html#Package-Dependencies

can be downloaded from the package web page[11] hosted on CRAN, the package source file is found under the downloads section.

```
Type: Package
Package: lubridate
Title: Make Dealing with Dates a Little Easier

Version: 1.8.0

Maintainer: Vitalie Spinu <spinuvit@gmail.com>

Depends: methods, R (>= 3.2)
Imports: generics
Suggests: covr, knitr, testthat (>= 2.1.0), vctrs (>= 0.3.0), rmarkdown
Enhances: chron, timeDate, tis, zoo
LinkingTo: cpp11 (>= 0.2.7)
```

Listing 1: Selected lines from the description text file for the **lubridate** R package. We are only showing the main fields and the list of package dependencies. There are additional fields that we have removed such as a description of the package.

The information provided in the description files for the packages is then used to help generate the CRAN package web pages (see, Wickham and Bryan 2015, section 8.2[12]). We can see on the package web page[11] how the information is displayed for the **lubridate** R package (Grolemund and Wickham, 2011). The package web pages include additional information that is not found in the description files, such as any Task Views that the package is allocated to.

The ways that packages communicate with each other can then be represented as a network, with edges denoting the type of relation between the nodes (the packages). Figure 1 shows the package directive network for the **cranly** R package (Kosmidis, 2019). The package directive networks are directed networks, with the direction of an edge defining which package "uses" another. For example, an imports edge from **visnetwork** (Almende B.V. et al., 2021) to **cranly** means **cranly** imports the **visnetwork** package.

[11] **lubridate** package web page: `https://cran.r-project.org/web/packages/lubridate/index.html`

[12] Section 8.2 from Wickham and Bryan (2015): `https://r-pkgs.org/description.html#description-title-description`

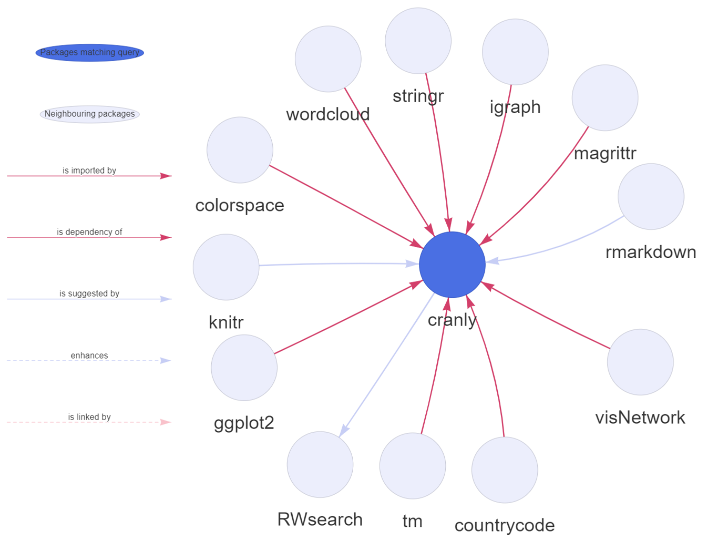

Figure 1: The package directive network for the **cranly** R package. Click on the figure to be sent to interactive version.
(Visit this web page[13] to view interactive figures for this report)

In addition to the package directives, we can also look at the CRAN package data through the perspective of the authors of the packages. We can view the collaboration between R package developers as a social network, with nodes representing authors and edges representing the packages that both authors worked on. Figure 2 shows the collaboration network of Patrice Kiener who is the author of the **RWsearch** R package (Kiener, 2021), can see from Figure 1 that **RWsearch** suggest the **cranly** R package.

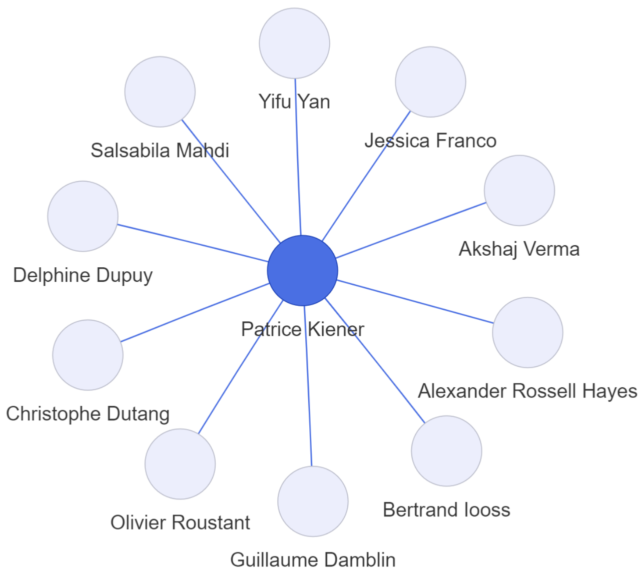

Figure 2: The author collaboration network for Patrice Kiener (author of the **RWsearch** R package).

[13]Web page for interactive figures for this report: https://dylandijk.github.io/Dissertation_Figures/index.html

1.3 cranly R Package

Before the package data can be analysed it must be extracted from CRAN. The metadata for R packages can be downloaded from CRAN using the `CRAN_package_db()` function from the **tools** R package (this is a base R[2] package, R Core Team 2020). This function loads the data for of all packages listed on CRAN[14]. Calling the function outputs a data frame object containing most of the metadata found in the description files for all of the current packages in the CRAN package repository (see, CRANtools help file[15] part of the **tools** R package help files[16]). This object contains many character strings, and needs to be cleaned in order to extract the information.

The **cranly** R package (Kosmidis, 2019) provides methods to clean this extracted data, so you are left with data that can be easily interpreted and manipulated. The package also offers methods to build both the package directives networks and collaboration networks using the information in the extracted data. The figures in the previous subsection (Figure 1 and Figure 2) where made using **cranly.**

Using **cranly**, a year worth of daily snapshots of the CRAN ecosystem has been collected by Ioannis Kosmidis. These snapshots started on the 11th of September 2019 and are still being captured (there have been some breaks).

There are some small issues with the data, not all data is perfectly extracted from CRAN. For example for the **igraph** R package (Csardi and Nepusz, 2006), the data extracted by **cranly** (Kosmidis, 2019) says that this package has no authors. This is because the authors have not been listed in the standard way on CRAN. Instead, they have linked to a separate text file[17] that breaks down the contribution of each author in further detail.

This problem exists for seven packages: **igraph, AnaCoDa, My.stepwise, RMOAjars, boilerpipeR, git2r** and **paletteer**. To overcome this we have used the maintainer of the packages as the nominated author.

Another issue with the data is the inconsistency in which authors name themselves on packages. For example there are cases where an author will include the initials for their middle name for a certain package and then elsewhere they will not. Figure 3 gives an example of an author using different variations of their name across packages in CRAN.

[14]List of packages available in CRAN: https://cran.r-project.org/web/packages/available_packages_by_name.html

[15]CRANtools is a subset of functions from the **tools** R package.
CRANtools help file: https://stat.ethz.ch/R-manual/R-devel/library/tools/html/CRANtools.html

[16]**tools** R package help files: https://stat.ethz.ch/R-manual/R-devel/library/tools/help/

[17]Text file provided in the author field on the **igraph** R package webpage: https://cran.r-project.org/web/packages/igraph/AUTHORS

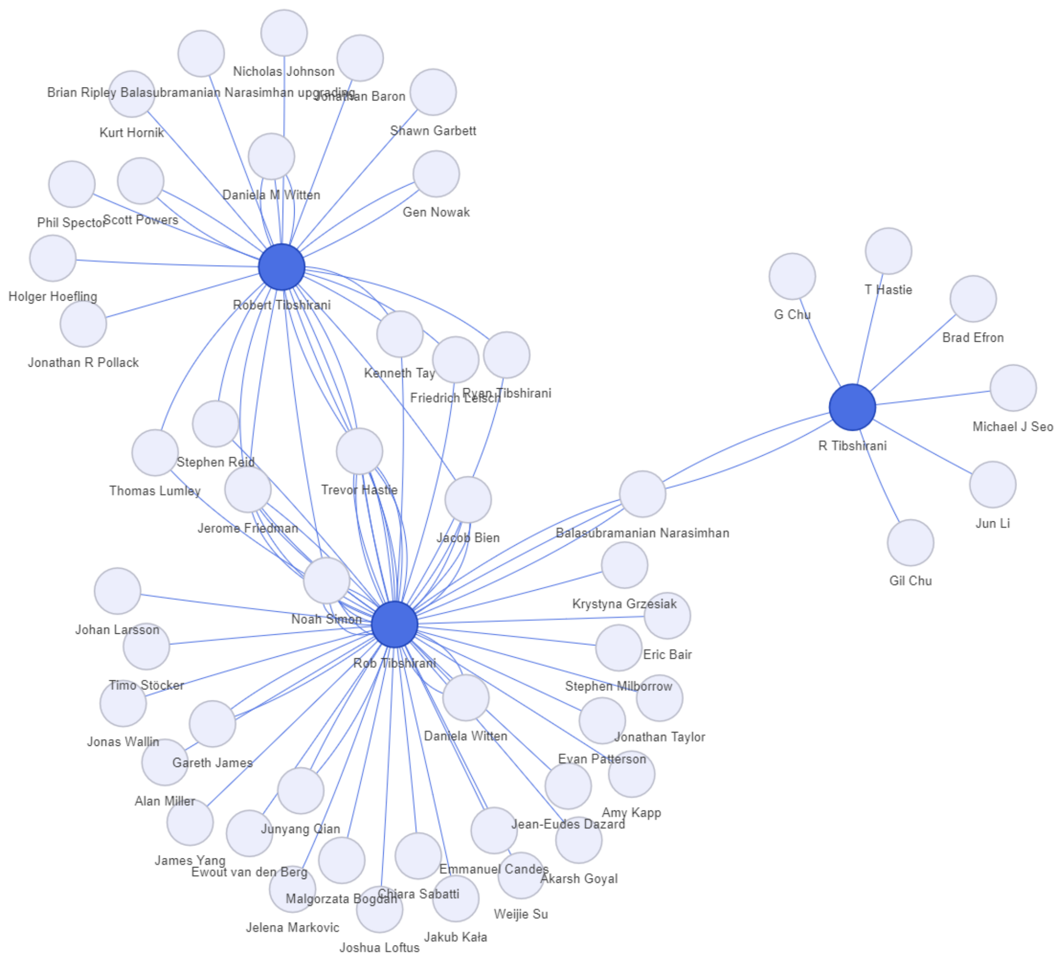

Figure 3: Author network for Robert Tibshirani (author of the **glmnet** R package). The figure shows that the package data extracted from CRAN represents these variations in names as individual authors.

In addition to **cranly**, another useful resource to observe the evolution of CRAN is the Microsoft R Application Network (MRAN[18]). MRAN hosts daily snapshots of the CRAN R packages and R releases as far back as the 17th of September 2014. These snapshot are taken at precisely midnight UTC using the `checkpoint-server`[19] which is the backend of the **checkpoint** R package (see the documentation[20] describing the workflow MRAN uses to capture the snapshots). The snapshots stored by Ioannis Kosmidis using **cranly** are taken between 15:00 and 17:00 UTC therefore there can be slight differences with the information stored by MRAN. The MRAN web page also has a CRAN Time Machine[21]

[18]Microsoft R Application Network: `https://mran.microsoft.com/`

[19]GitHub repository for the `checkpoint-server`: `https://github.com/RevolutionAnalytics/checkpoint-server`

[20]MRAN snapshots workflow: `https://mran.microsoft.com/documents/rro/reproducibility#snapshots`

[21]CRAN Time Machine: `https://mran.microsoft.com/timemachine`

web application that allows you to browse easily through the daily snapshots.

1.4 Task Views

1.4.1 Description of the CRAN Task View Initiative

The CRAN package ecosystem being as large as it is, motivates the need to organise the packages so that users can find the package they need. The strategy that CRAN is implementing in order to solve this problem, is through the CRAN Task Views[22]. These are groups of packages organised in terms of relevance for tasks in a certain topic. For example the **lubridate** R package (Grolemund and Wickham, 2011) belongs to the ReproducibleResearch Task View[23] and also the TimeSeries Task View[24] .

The individual Task Views are maintained by volunteers, and the Task View management is overseen by the CRAN Task View editors (Achim Zeileis, Roger Bivand, Dirk Eddelbuettel, Rocío Joo, Nathalie Vialaneix and David Meyer). The aim of the Task Views is to provide guidance and they are not meant to endorse the "best" packages for a given task. The views are intended to have a sharp focus so that it is sufficiently clear which packages should be included or excluded (see, the CRAN Task View Initiative GitHub repository README file[25]).

1.4.2 Transfer of the CRAN Task View Infrastructure to GitHub

Towards the end of 2021, the CRAN Task View editors started to transfer the infrastructure of the development and maintenance of CRAN Task Views from R-Forge to GitHub[26,27]. During this process they converted XML files, that created the web pages for each of the Task Views, into Markdown files. For example the TimeSeries Task View repository on GitHub contains a Markdown file[28] that generates the web page for the TimeSeries Task View[24]. These source files were originally stored in R-Forge, which is a central platform for the development of R packages.

[22]CRAN Task Views: https://cran.r-project.org/web/views/

[23]ReproducibleResearch Task View web page: https://cran.r-project.org/web/views/ReproducibleResearch.html

[24]TimeSeries Task View web page: https://cran.r-project.org/web/views/TimeSeries.html

[25]CRAN Task View Initiative GitHub repository README file: https://github.com/cran-task-views/ctv#cran-task-view-initiative-

[26]R-Forge CRAN Task Views home page: https://ctv.r-forge.r-project.org/index.html

[27]Link to change-log in R-Forge: https://ctv.r-forge.r-project.org/news/index.html#ctv-0-9-0-2021-12-14

[28]Markdown file that generates the TimeSeries web page: https://github.com/cran-task-views/TimeSeries/blob/main/TimeSeries.md

By the end of January 2022 the conversions were completed and the Markdown files were placed into individual repositories for each Task View, within the CRAN Task Views GitHub account[29]. All of the repositories were not made immediately public, as maintainers were still making amendments to the files. The CRAN Task Views account also contains the ctv[30] repository.

During the transfer process a few Task Views were retired and were archived in R-Forge. For example, the Graphics Task View was removed as it was thought to be too broad too be useful and had not been maintained for a while. The transition to GitHub was then finalised and made public on the 25th of March 2022 (see the 92nd commit[31] on the ctv GitHub repository). At this date there were 36 Task Views in total, compared to the 4th of November 2021 where there were 41 Task Views.

The updated Task View source files were then updated onto the CRAN Task Views[22] web page on the 4th of April 2022 (see the announcement by Achim Zeileis via this thread[32] on Twitter). Once the new CRAN Task View workflow had been launched the Task View editors then started to look at new Task View proposals (see this comment[33] on a discussion about the proposal of a SportsAnalytics Task View).

1.4.3 Storing Task View Snapshots

During my project I have followed this transition, and paid attention to the issues raised by Task View editors on the repository. The problems discussed by the editors can be found on the open and closed sections within the issues tab[34] of the ctv GitHub account.

Additionally, I have stored Task View snapshots as different Task Views were removed using the `tvdb_down()` function from the **RWsearch** R package (Kiener, 2021). The `tvdb_down()` function downloads information of the Task Views at the current day. This data is stored as a list of length given by the number of Task Views at that time of running the function. The package also provides other functions that allow us to extract information from the snapshots. The `tvdb_vec()` function lists the names of all the Task Views in the loaded snapshot, and `tvdb_pkgs()` takes a Task View name as an argument

[29]CRAN Task Views GitHub account: https://github.com/cran-task-views

[30]CRAN Task View (ctv) Initiative GitHub repository: https://github.com/cran-task-views/ctv

[31]Announcement of completion of transfer to GitHub from R-Forge: https://github.com/cran-task-views/ctv/commit/bda537c5d496ad7ee042f77f07a4d1a90d63a1a7

[32]Announcement of relaunch of CRAN Task Views on the 4th of April: https://twitter.com/AchimZeileis/status/1510945091980038145

[33]SportsAnalytics Task View proposal: https://github.com/cran-task-views/ctv/issues/11#issuecomment-1094135596

[34]ctv repository within the CRAN Task Views GitHub account - Issues tab: https://github.com/cran-task-views/ctv/issues

and outputs a character vector of all the package assigned to that Task View.

As well as storing snapshots, the CRAN Time Machine[21] described at the end of Section 1.3 is useful to follow how the Task Views have changed over time. Using this application we can look at the evolution of the Task Views by selecting a date and then clicking on the Task Views tab on the CRAN Time Machine viewing window.

For the rest of this report we use two snapshots of CRAN and its Task Views. We use a snapshot taken on the 20th of February 2022 before the transfer process was completed. Then a snapshot taken on the 4th of April 2022 when the new workflow was launched. In the Appendix, Table 3 and Table 4 give summary statistics of each snapshot.

In Section 2, we present results from an exploratory data analysis using these two snapshots. We then compute network statistics of the Task View author subnetworks using the snapshot taken on the 4th of April 2022.

In Section 3 we create a similarity measure between Task Views using the 20th of February 2022 snapshot and compare this measure with the decisions made by the CRAN Task View editors during the transfer process. We then create a final similarity measure with the 4th of April 2022 snapshot, when the transfer of the Task View management to GitHub was completed. In Section 4 we train the model using the snapshot taken on the 4th of April 2022.

1.5 Notation

We now introduce mathematical notation to describe the package directive networks and the subnetworks formed by the Task Views.

As discussed in Section 1.2, the packages hosted on CRAN can be viewed as a network. The nodes of this network represent the packages, the edges between them the dependencies (see Figure 1 for example). We also covered the different package dependency types, and for the rest of this report we just use the **Hard Dependencies** between packages and ignore the **Soft Dependencies**.

We denote the package dependency network of packages hosted on CRAN by the directed graph object $G(V, E)$, which is defined by the node set V and the directed edge set E. We let N denote the number of nodes in set V and L the number of directed edges in E.

The set E has directed edges as elements, where a directed edge $e_k \in E$ is defined as an ordered pair (v_i, v_j) with $v_i, v_j \in V$. For the directed edges, we refer to the first element of the pair as the **tail** and the second element as the **head** of the edge.

The set of edges E can be encoded into an adjacency matrix A, with elements $A_{ij} = 1$ if there exists an edge from node i to node j, and zero if not. It follows that A has dimension $N \times N$. For directed networks the adjacency matrix can be asymmetric.

Now in order to describe the Task View subnetworks, we let V_k denote the set of nodes (CRAN packages) belonging to Task View k and let K be the number of Task Views. Each Task View consists of packages that are hosted on CRAN, and therefore $V_k \subset V$ $\forall k \in \{1, \ldots, K\}$.

We define the subnetwork for Task View k as the induced subgraph of G generated by the node set V_k. An induced subgraph of a graph G generated by a node set W, is defined as the Graph whose node set is W and the edge set is all the edges in E that have both their tail and head in W.

In addition, in Section 1.2 we also mentioned that the author data of the packages hosted on CRAN can be viewed as a network (see Figure 2 for example). The nodes of this network represent the authors, and the edges between them the collaboration of the authors.

We denote the author collaboration network by the undirected graph object $G'(V', E')$, which is defined by the node set V' and the edge set E'. E' is an undirected edge set, where an edge $e'_k \in E'$ is an unordered pair (v'_i, v'_j) with $v'_i, v'_j \in V'$.

Similar to the package directive notation, we let V'_k denote the set of authors who developed a package (hosted on CRAN) that is assigned to Task View k. V' is the set of all authors who have developed a package in CRAN and therefore $V'_k \subset V'$ $\forall k \in \{1, \ldots, K\}$.

I define the author collaboration network for Task View k, to be the induced subgraph of G' generated by node set V'_k. It is important to note that by this construction the subgraph can include edges that represent packages that are not allocated to Task View k.

2 Exploratory Analysis of the Task Views

2.1 Aim

In this section we present the results of an exploratory data analysis that provides an overview of the data. This includes a preliminary analysis and then a network analysis of the collaborative author networks formed within the Task Views.

A social network is defined as is a collection of people, each of whom is acquainted with some subset of the others (see M. E. J. Newman, 2001b). Therefore the collaboration network of package developers is a social network. In the literature there are many case studies analysing scientific collaboration networks (see M. E. J. Newman, 2001b; M. E. J. Newman, 2001a; Bella, Gandullia, and Preti, 2021). They explore ideas such as the connectivity of the networks and the influence of particular nodes in the network, we take these ideas and implement them onto the CRAN collaboration network.

From the analysis, we can discover the influential package developers according to the statistics we calculate. Throughout the analysis, we define the different network statistics and display the results.

2.2 Preliminary Analysis

2.2.1 CRAN on the 20th of February 2022

Before analysing the structure of the networks, it is good to get an overview of the Task Views. We use a snapshot taken on the 20th of February 2022 before the transfer was completed, and then a snapshot taken on the 4th of April 2022 when the transfer was completed and uploaded on to CRAN. In the Appendix we provide tables with the results.

On the 20th of February 2022 date there were 18966 packages hosted on CRAN (see Table 3). 3425 of these packages were assigned to at least one Task View, and hence there were 15541 packages that had no assigned Task View. Figure 4 shows the number of packages in each of the 40 Task Views, and the number of authors who developed at least one package in each of the Task Views. The values shown in Figure 4 are provided in Table 5 (the 20th of February 2022 columns), in the Appendix.

If we look at the entire CRAN network, the median number of packages developed by each author was 1. This varies however if we look at authors within Task Views, for example, the TeachingStatistics Task View[35] had a median of 4. The distributions of the number of packages developed by authors is very positively skewed with most Task Views consisting of a small group of authors that have developed a large amount of packages. In the TeachingStatistics Task View[35] there are 6 authors that have developed over 50 packages in CRAN.

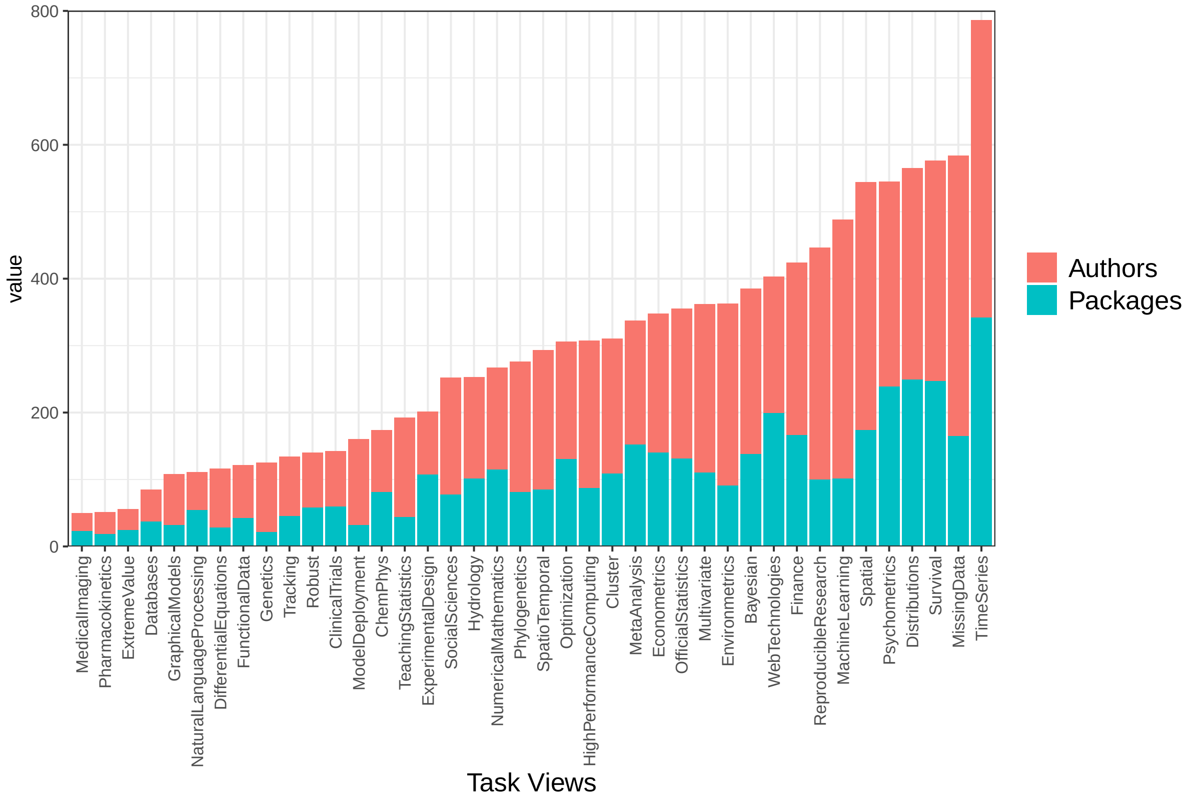

Figure 4: Number of packages and authors in each of the Task Views (20th of February 2022).

If we look at the average number of authors that worked on the development of packages, this is again very skewed with the median number of authors being 2 for the whole CRAN network. This is mostly true for the number of authors of Task View packages, with the medians also being equal to 2. An example of some packages that have a large number of developers are: **mlpack** from the MachineLearning Task View[36] with 133 authors, and **knitr** from the Reproducible Research Task View[23] with 100 authors.

At this date we can also look at the overlap of packages in the different Task Views. Figure 5 shows the proportion of packages that exist in both Task Views.

[35]TeachingStatistics Task View web page: https://cran.r-project.org/web/views/TeachingStatistics.html

[36]MachineLearning Task View web page: https://cran.r-project.org/web/views/MachineLearning.html

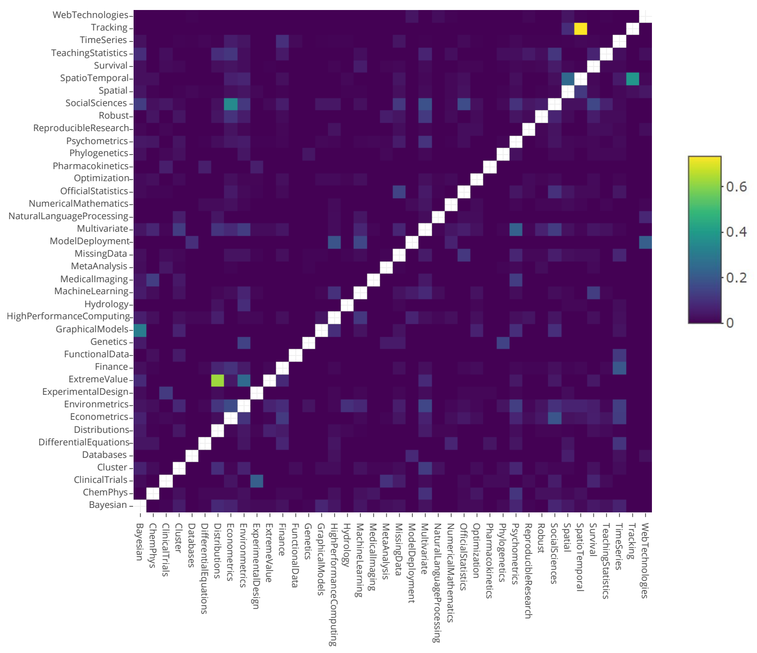

Figure 5: (x, y) element is the number of packages in both Task View x and y divided by the number of packages in task view y (20th of February 2022).

Table 1 shows the the top five pairs of packages with highest overlap values. We can see that there is a very high proportion (73.3%) of packages that belong to the Tracking Task View[37] that also belong to the SpatioTemporal Task View[38].

Top 5 pairs with largest overlap proportions					
x y	SpatioTemporal Tracking	Distributions ExtremeValue	Tracking SpatioTemporal	Econometrics SocialSciences	Bayesian GraphicalModels
	0.733	0.625	0.388	0.35	0.313

Table 1: Top 5 largest overlap proportions. Proportions are calculated in same way as Figure 5 with the x Task View in top row and y Task View in bottom row.

2.2.2 cranlogs R Package

Additionally to getting an overview on the number of packages and authors, we can use the **cranlogs** R package (Csárdi, 2019) to get an idea of how frequently packages are downloaded by users. The **cranlogs** R package provides an API to the database of CRAN

[37]Tracking Task View web page: https://cran.r-project.org/web/views/Tracking.html

[38]SpatioTemporal Task View web page: https://cran.r-project.org/web/views/SpatioTemporal.html

package download counts from the RStudio CRAN mirror (one of the mirror servers in CRAN).

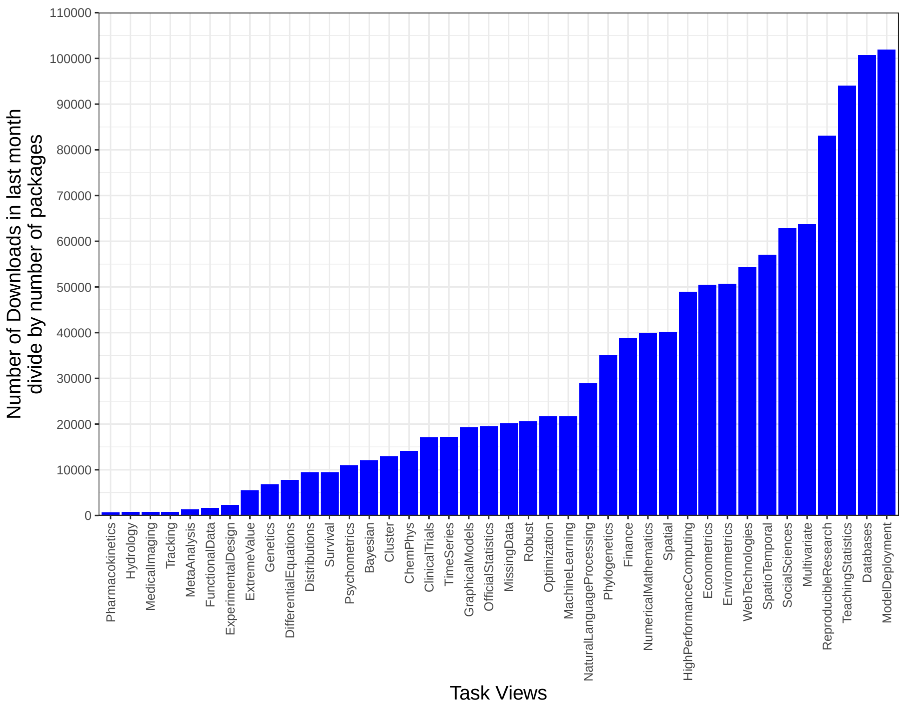

Figure 6: Number of downloads of packages belonging to each Task from the 20th of January 2022 to the 20th of February 2022 View, divided by the number of packages in each Task View (20th of February 2022).

Using **cranlogs** we can uncover which Task Views are most popular, Figure 6 shows that the ModelDeployment[39] and Databases Task View[40] were the most popular according to monthly downloads.

[39]ModelDeployment Task View web page: `https://cran.r-project.org/web/views/ModelDeployment.html`

[40]Databases Task View web page: `https://cran.r-project.org/web/views/Databases.html`

2.2.3 CRAN on the 4th of April 2022

At this date there were 36 Task Views in comparison to the 40 on the 20th of February (see Table 4). However the number of packages that were assigned to a Task View increased by 91 packages.

The four Task Views that were removed are: **Genetics, Multivariate, Phylogenetics** and **SocialSciences** (these are highlighted in red in Table 5). The values given in Figure 4 and Figure 6, for the Task Views that were not archived, do not change substantially between the two dates.

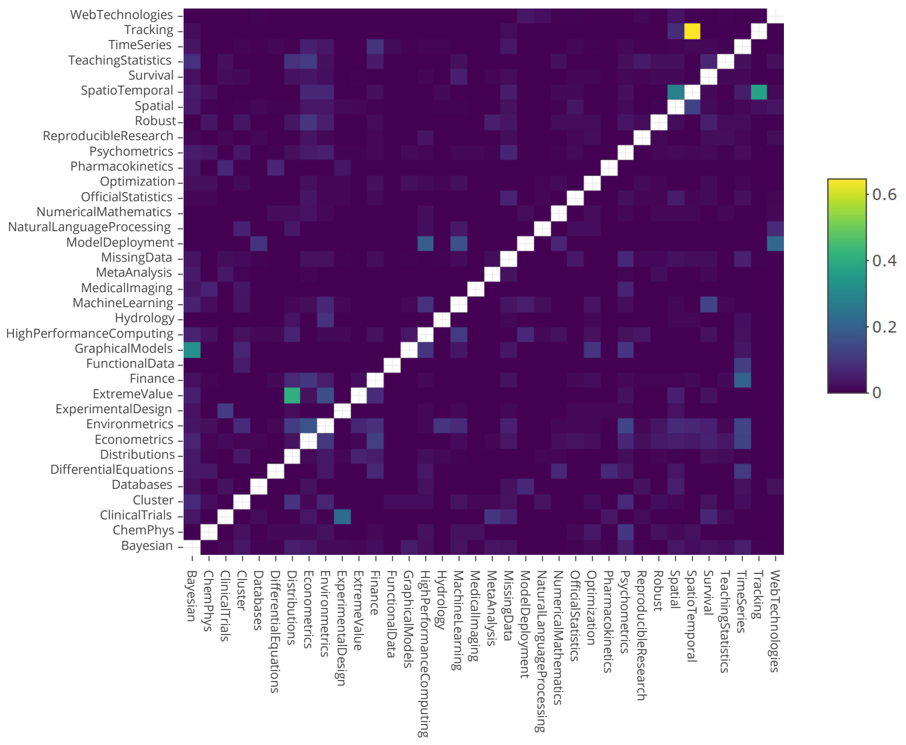

Figure 7: (x, y) element is the number of packages in both task view x and y divided by the number of packages in task view y (4th of April 2022).

Comparing Figure 5 and Figure 7, we can see that in general there is a reduction in overlap of packages between the two dates. For example the overlap between the Distributions[41] and Extremevalue Task View[42] dropped from 63% to 41%.

[41]Distributions Task View web page: https://cran.r-project.org/web/views/Distributions.html

[42]Extremevalue Task View web page: https://cran.r-project.org/web/views/ExtremeValue.html

2.3 Connectivity of the Author Subnetworks

In this subsection we look at how connected the authors of the Task Views are, by utilizing the collaboration networks of the packages.

In Joo et al. (2020) the maintainers of the Tracking Task View highlighted the issue of fragmentation in the package dependency network of their Task View. They made the point that "many of these tools have proliferated in isolation, making it challenging for users to select the most appropriate method for the question in hand". They examined the dependency structure of packages, which explicitly determines whether a package is enhancing an existing package or if it has been developed in isolation.

Having a connected collaboration network, allows for information and ideas to pass easily between developers. In regards to developing new packages this is very important, if developers are aware of the work that is taking place it is less likely that redundant packages will be developed. This could be the case however if the collaboration network is very fragmented. Of course, having worked on a package together does not determine perfectly the acquaintance between two people, and we are just using it here as a proxy.

One technique we can utilize to measure the connectedness of the Task View subnetworks is by looking at the size of the largest component. We introduce some further notation and define the largest component.

Definition 1. Path (Antiqueira and Fontoura Costa, 2009). A path $p(i,j)$ from node i to node j is denoted by a sequence of neighboring nodes: $p(i,j) := (v_1, v_2, ..., v_m, v_{m+1})$ where $v_1 = i$, $v_{m+1} = j$ and $A_{v_i,v_{i+1}} = 1 \quad \forall i \in \{1,\dots,m\}$. The length of the path $p(i,j)$ is $\omega(p(i,j)) = m$, this is the the number of edges along the path.

Definition 2. Cardinality. The cardinality of a set is the number of elements in that set.

Definition 3. Component. A component is a subset of the nodes of a network such that there exists at least one path from each member of that subset to each other member, and such that no other node in the network can be added to the subset while preserving this property (definition from M. Newman, 2018, Section 6.12). The **largest component** is the component with the largest cardinality.

The largest component is sometimes referred to as the giant component, however "giant component" has a specific meaning in network theory and is not precisely synonymous with "largest component." (see M. Newman, 2018, Section 10.1 and 11.5).

We calculate the size of the largest components for the collaborative subnetworks of the Task Views. As noted previously the edges of the author Task View subnetworks denote packages that the adjacent authors both worked on, but we are not restricting these packages to be assigned to the selected Task View (we define the induced subgraph in Section 1.5). We want to analyse the "connectivity" of the authors, therefore it would not make sense to restrict the edges.

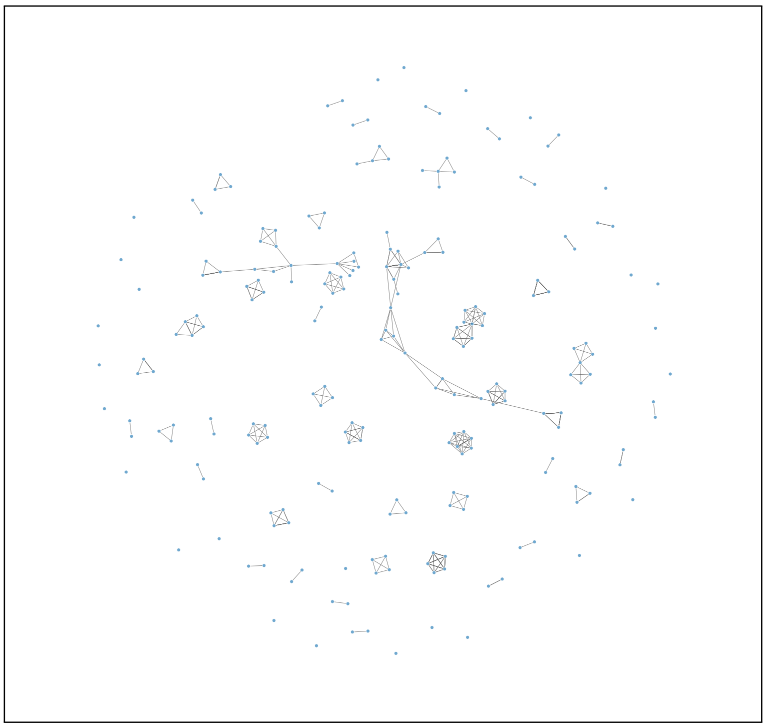

Figure 8: Author network of authors who have published packages belonging to the ExperimentalDesign Task View. This is the Task View with the smallest largest component (see Definiton 3) as a proportion to the total number of authors.

Table 6 gives the largest components of the author networks for each Task View as a proportion of the total number of authors. The mean proportion of the giant components is 64% (with SD of 22%). The largest proportion belongs to the ModelDeployment Task View[39] with a value of 95% and the minimum value belongs to the ExperimentalDesign Task View[43] with a value of 11%.

Figure 8 shows the author network for the ExperimentalDesign Task View[43] (this figure was made using the **networkD3** R package (Allaire et al., 2017) within the modified **cranly** plot function).

Another aspect of the author network to investigate is the average geodesic distance between pairs of authors, this would give an idea of the "distance" between connected authors. We define the geodesic distance in Definiton 4.

[43]ExperimentalDesign Task View web page: `https://cran.r-project.org/web/views/ExperimentalDesign.html`

Definition 4. Geodesic Distance (Antiqueira and Fontoura Costa, 2009). The shortest path length between two nodes i and j is defined as: $s(i, j) := \min\{\omega(p(i, j))\}$. This is known as the **geodesic distance** between the nodes (see M. Newman, 2018, Section 6.11.1 for more details).

However, the geodesic distance between unconnected authors will be undefined and therefore we need to compute these average distances on a component of the subgraphs. We therefore compute the geodesic distances pairwise for all authors in the largest component of the Task View subnetworks.

To compute the pairwise geodesic distances we use the `distances()` function from the **igraph** R package (Csardi and Nepusz, 2006). We then take the average of these distances for each subnetwork, these values are displayed in Table 6.

2.4 Influence of the Authors

Another idea we can investigate using the network data is to measure the influence of nodes in the network. For the collaboration network we can look at which authors are important in connecting different groups of the network.

One of the statistics we can consider is the betweenness of a node, this looks at the number of shortest paths that pass through the node. This measure of centrality can be interpreted as the control an author has on the information travelling between others (see M. E. J. Newman, 2001a, Section B). It was introduced by Freeman (1977), and we use some notation from Kolaczyk and Csárdi (2014, Section 4.2.2) to define it in Definiton 5.

Definition 5. Betweenness.

$$c_B(\nu) = \sum_{s \neq t \neq \nu \in V} b_{s,t}(\nu) \tag{1}$$

$$b_{s,t}(\nu) = \begin{cases} 0 & \text{if there exists no path between nodes } s \text{ and } t \\ \frac{\sigma(s,t|\nu)}{\sigma(s,t)} & \text{otherwise} \end{cases}$$

Where $\sigma(s,t)$ is the number of shortest paths from s to t, and $\sigma(s,t|\nu)$ is the number of shortest paths between s and t that pass through the node ν. In equation (1) we are summing over all nodes s and t, apart form ν, that are different from each other. If for each pair of s and t there only exists one shortest path, then $c_B(\nu)$ will just count the number of shortest paths going through ν.

$b_{s,t}(\nu)$ can be thought of as the probability that point ν falls on a randomly selected geodesic linking s with t.

The value of $c_B(\nu)$ depends on the number of points in the graph. Freeman (1977) showed that the maximum possible value of $c_B(\nu)$, for any node ν, for any graph with n nodes is given by equation (2).

$$\frac{n^2 - 3n + 2}{2} \tag{2}$$

Therefore using (2) we can get a normalised betweenness score given by equation (3). Values computed from this normalised version can now be compared across the Task View subnetworks.

$$c'_B(\nu) = \frac{2\, c_B(\nu)}{n^2 - 3n + 2} \tag{3}$$

The normalised version of the betweenness centrality measure can be computed with the **igraph** package (Csardi and Nepusz 2006, see betweenness documentation[44], in particular the description of the normalized argument).

In Table 7 we show the top five authors with highest betweenness values. The largest value across Task Views belongs to Dirk Eddelbuettel in the MachineLearning Task View[36] with a normalised betweenness score of 0.38.

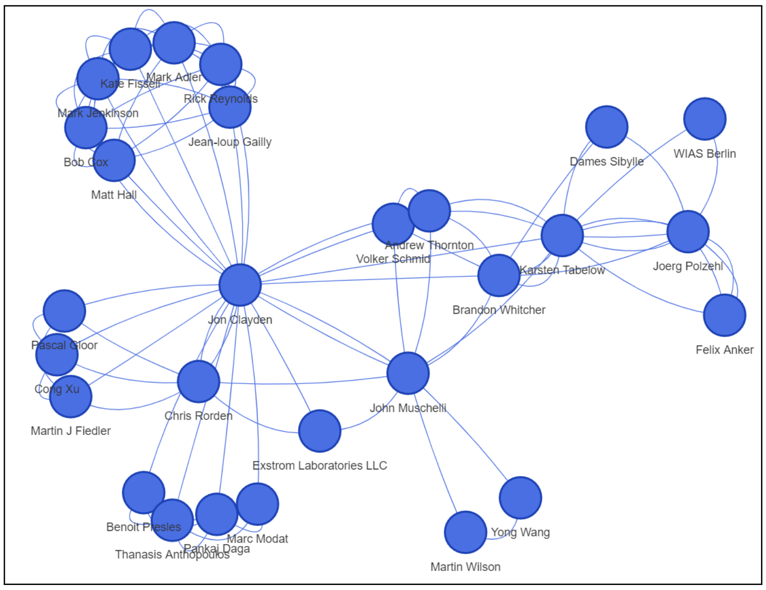

Figure 9: Largest component of the author network of authors who have published packages belonging to the MedicalImaging Task View.

Figure 9 shows the largest component of the MedicalImaging Task View[45]. For this Task View Jon Clayden is ranked the highest (see Table 7), from Figure 9 can see that Jon Clayden connects three clusters of authors.

[44]Vertex and edge betweenness centrality documentation for the **igraph** package: `https://igraph.org/r/doc/betweenness.html`

[45]`https://cran.r-project.org/web/views/MedicalImaging.html`

3 Similarity of Task Views

3.1 Aim

As described on the CRAN Task View initiative GitHub page, the purpose of the Task Views is to provide a sharp focus on the packages that are relevant for a task. The views are intended to have a sharp focus so that it is sufficiently clear which packages should be included or excluded. The editors want to avoid overlap between Task Views. Therefore, it is important that no two Task Views are too similar.

The CRAN Task View editors have to make decisions on which Task Views to create and whether they need to remove an existing Task View. Therefore a measure of how similar two Task Views are, would help make such decisions.

In this section we utilise the package dependency information, and how the subnetworks formed by the Task Views interact with each other to create a measure of similarity between any two Task Views. A similarity measure is a real-valued function that quantifies the similarity between two objects, with larger values implying that the objects are more similar.

In addition to the dependencies between packages, there is text data describing the Task Views. In Section 3.3 we incorporate this information into the similarity measure.

3.2 Measuring Similarity of Task Views using Package Dependencies

3.2.1 Defining the Similarity Measure

First of all we will define a measure of similarity between Task Views using the dependency structure of the packages.

The way we have decided to carry this out is by looking at the number of hard package dependencies (see Section 1.2) that travel across the Task Views relative to the number of hard package dependencies within the Task Views. The reason for this choice is because a package that imports another package (for a specific task) is likely to be used to solve similar problems.

To be more specific, given two Task Views we have defined the similarity measure between them as the ratio of the number of **between** and **within** edges of the Task Views. This is given in equation (4), where V_k and V_s are the node sets of Task Views k and s respectively as defined in Section 1.5.

$$s(V_s, V_k) \;=\; \frac{\text{number of edges between } \; V_k \; \text{ and } \; V_s}{\text{number of edges within } V_s \;\; + \;\; \text{number of edges within } V_k} \tag{4}$$

The **within** edges are edges whose head and tail nodes (defined in Section 1.5) both belong to the same Task View and neither of them are assigned to both Task Views. The **between** edges are edges whose head and tail nodes belong to different Task Views. If any of the tail or head nodes belong to both Task Views then the edge will be classified as a between edge. Existence of a between edge implies a package has a direct dependency on a package from the other Task View.

To then compute the similarity value we divide the number of between edges by the total number of within edges. As we are summing edges with disregard to the direction of them, the similarity measure is symmetric.

From the way we have constructed $s(V_s, V_k)$, for any two different Task Views with corresponding node sets V_s and V_k, $s(V_s, V_k)$ takes values in the interval $[0, \infty)$. If there exists no edges within both of the Task Views then it is undefined, however this not the case for any of the Task Views. This does mean that the similarity of a Task View to itself will be undefined, but we are not interested in measuring the similarity between identical Task Views. Additionally it is highly unlikely a Task View will be proposed that contains the identical packages to an existing Task View.

If $s(V_s, V_k)$ equals zero for a pair of Task Views this then implies that there are no edges travelling from one Task View to another (there are no between edges).

Equation (4) can be written using the adjacency matrix A for the entire CRAN package network and vectors that take values of zero or one:

$s(V_s, V_k) \;=$

$$\frac{\mathbf{1}^T_{V_s \cap V_k^c} A\, \mathbf{1}_{V_s \cap V_k} \;+\; \mathbf{1}^T_{V_s \cap V_k} A\, \mathbf{1}_{V_s \cap V_k^c} \;+\; \mathbf{1}^T_{V_k \cap V_s^c} A\, \mathbf{1}_{V_s \cap V_k} \;+\; \mathbf{1}^T_{V_s \cap V_k} A\, \mathbf{1}_{V_k \cap V_s^c} \;+\; \mathbf{1}^T_{V_s \cap V_k} A\, \mathbf{1}_{V_s \cap V_k} \;+\; \mathbf{1}^T_{V_s \cap V_k^c} A\, \mathbf{1}_{V_s^c \cap V_k} \;+\; \mathbf{1}^T_{V_k \cap V_s^c} A\, \mathbf{1}_{V_k^c \cap V_s}}{\mathbf{1}^T_{V_k \cap V_s^c} A\, \mathbf{1}_{V_k \cap V_s^c} \;+\; \mathbf{1}^T_{V_s \cap V_k^c} A\, \mathbf{1}_{V_s \cap V_k^c}}$$

V_k^c denotes the complement of V_k with respect to the node set V (V is the set of all packages in CRAN where we let N be the number of packages, defined in Section 1.5). V_k denotes the node set for the Task View k and therefore V_k^c is the set of nodes in CRAN that do not belong to to Task View k. In this formula we use node set intersections, for example $V_k \cap V_s^c$ is the set of packages that are in Task View k but not in Task View s.

In this formula we use the notation $\mathbf{1}_B$ for some node set $B \subset V$, to define a column vector of size N which takes value 1 at position i if node i of the node set V belongs to B and zero otherwise. Therefore for example $\mathbf{1}^T_{V_s \cap V_k^c} A$ gives a N dimensional row vector with the ith element being the number of edges that travel from the node set $V_s \cap V_k^c$ to node i. Consequently $\mathbf{1}^T_{V_s \cap V_k^c} A \, \mathbf{1}_{V_s \cap V_k}$ will give the number of edges that travel from the node set $V_s \cap V_k^c$ to the node set $V_s \cap V_k$.

The subtlety here is when the head or tail belong to both Task Views, in this case I have still classified this as a between edge. For example, if package 1 belongs to “Bayesian” and package 2 belongs to “Bayesian” and “HighPerformanceComputing” then the edge from package to 1 to 2 will be a between edge and vice versa. Therefore two Task Views that have a large overlap in packages is more likely to have a large value according to this similarity measure based on package dependencies.

A large value in this measure implies that there are many package dependencies connecting packages from one Task View to another, and therefore shows that the packages in the different Task Views are reliant on each other. Also, as we are dividing by the number of edges within Task Views this normalises for the variation in the number of packages in each of the Task Views. Additionally it normalises for Task Views that contain packages which have a large number of dependencies.

3.2.2 Examining Results

We have calculated equation (4) for all possible pairs of Task Views using the snapshot taken on the 20th of February 2022. At this date there were 40 Task Views so we get a 40×40 matrix. This date was before the final transfer to GitHub was completed and hence this includes a few Task Views that had been retired during the transfer process. The heatmap for this similarity matrix is given in Figure 10.

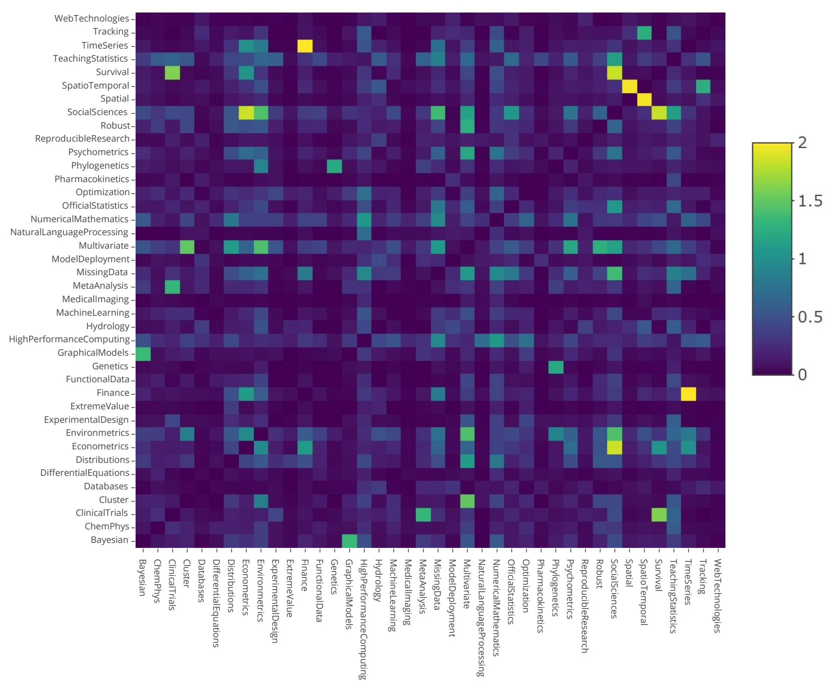

Figure 10: Similarity matrix constructed by computing the similarity measure given by equation (4) pairwise on the Task Views. Using the Task View snapshot taken on the 20th of February 2022.

Table 2 below shows the top five pairs of packages with highest values calculated from equation (4).

Top 5 pairs with largest similarity values				
TimeSeries Finance	Spatial SpatioTemporal	Econometrics SocialSciences	SocialSciences Survival	ClinicalTrials Survival
2.00	1.97	1.84	1.83	1.61

Table 2: Top 5 largest values from applying similarity measure using package dependencies (equation (4)) using snapshot of CRAN and its Task Views taken on the 20th of February 2022.

We mentioned that Task Views with large overlap in packages are likely to have a high value according to the similarity measure based on package dependencies (equation (4)). If we compare Table 1 and Table 2 we can see that the only shared pair is (Econometrics, SocialSciences). Therefore this similarity measure is not dominated by overlapping Task Views.

In Section 1.4 we highlighted the retirement of certain Task Views during the transfer process to GitHub. Using the similarity measure defined by equation (4) we can compare how our our analyses of similar Task Views compares to the decision made by the Task View maintainers. To do this we perform clustering using the package dependency similarity measure with the 20th of February snapshot to reveal any groupings of similar Task Views.

In order to use hierarchical clustering we first need to convert the similarity measure to a dissimilarity measure. If we multiply the values in the similarity matrix by -1 and add the maximum similarity value we then obtain a dissimilarity matrix that gives positive values for all the Task Views. With this dissimilarity measure we can then perform agglomerative hierarchical clustering with complete-linkage. This method of clustering is described in Algorithm 1 below (notation from, Shahin Tavakoli 2017, Multivariate Statistics notes, section 7.2).

Algorithm 1. Agglomerative hierarchical clustering

Let $o_1, \ldots, o_n$ be the objects to be clustered. Denote the initial clusters by:

$$C_1^{(0)} := \{o_1\}, \ldots, C_n^{(0)} := \{o_n\}$$

Set $D^{(0)}$ to be the distance matrix between the initial clusters $\{C_i^{(0)}\}_i$

For $k = 1, 2, \ldots, n-1$

a) Find $i \neq j$ such that $(D^{k-1})_{ij}$ is minimal.
Denote this distance by d^k

$$d^k := \min_{i \neq j} D_{ij}^{(k-1)}$$

b) Merge the two closest clusters $C_i^{(k-1)}$, $C_j^{(k-1)}$ together. Therefore you are then left with $n-k$ clusters $C_1^{(k)}, \ldots, C_{n-k}^{(k)}$, from these compute the pairwise distances between them to create the new distance matrix $D^{(k)}$.

Each element first starts out as an individual cluster, then clusters that are closest together are merged. After step 1 of the algorithm, part b) requires you to compute the distance between clusters that have more than one element. The method to compute the distance between clusters is called the linkage method. I have used the complete-linkage method, which takes the distance between two clusters to be the farthest elements in those clusters. Therefore the new distance matrix $D^{(k)}$ is calculated in the following way:

$$D_{ij}^{(k)} = \max_{a \in C_i^{(k)}, b \in C_j^{(k)}} D_{ab}^{(0)}$$

After carrying out this algorithm, we can display the construction of the clusters by a dendrogram. The dendrogram displays the order in which the clusters were combined. The horizontal lines display the distance between the clusters when they were merged, these are the $d^{(k)}$ values from Algorithm 1. Figure 11 below shows the dendrogram produced from carrying out the hierarchical clustering on my dissimilarity matrix.

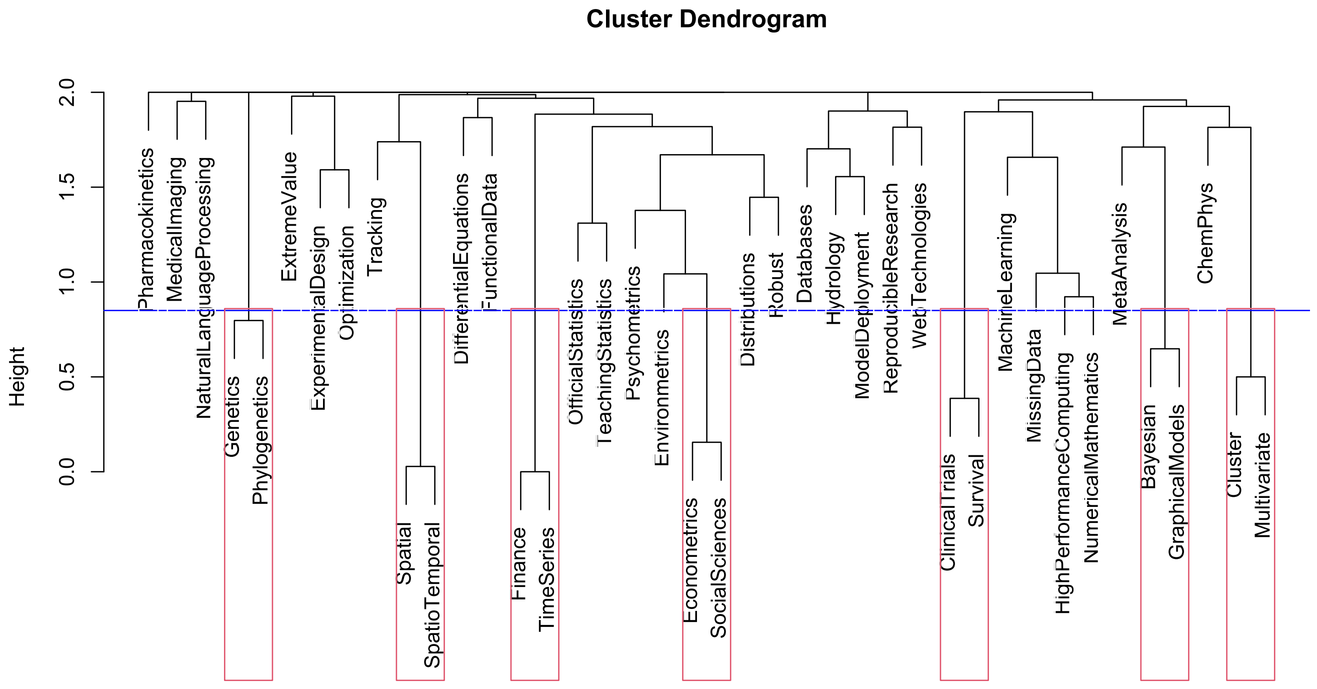

Figure 11: Dendrogram created from applying agglomerative hierarchical clustering with complete-linkage. Using the dissimilarity matrix constructed by multiplying the values in the similarity matrix (Figure 10) by -1 and adding the maximum similarity value.

If we set the threshold to 0.85 we retrieve seven clusters of pairs (see Figure 11): (Finance, TimeSeries), (Spatial, SpatioTemporal), (Econometrics, SocialSciences), (ClinicalTrials, Survival), (Cluster, Multivariate), (Bayesian, GraphicalModels) and (Genetics, Phylogenetics).

3.2.3 Comparing Results to the Decisions Made by the CRAN Task View Editors

Now if we look at the decisions made by the Task View maintainers between 20th of February and the 4th of April 2022, which we can follow from the open and closed sections within the issues tab[34] of the ctv GitHub account. We can see that the following Task Views were retired.

The **SocialSciences** Task View was one of the old existing task views that was useful when it was introduced but turned out to be too broad to be useful. Achim Zeileis and John Fox decided it would be better to separate this Task View into smaller topics, such as: mixed-effects models, social network analysis, demography, and causal analysis or matching/propensity scores (see issue 8[46] from the issues tab for the full discussion). This decision has led to the proposal[47] of a **CausalInference** Task View.

The **Multivariate** task view had been unmaintained for a long time. It was also a task view that was useful when it was established but turned out to be too broad to be maintainable. The Task View editors had made attempts in the past to find new maintainers that could give the Task View a sharper focus and carry on the maintenance. It was decided that this Task View should be retired (see issue 9[48]from the issues tab for the full discussion).

The maintainer of the **Genetics** Task View stepped down. This was a small task view which had not been updated for a while, hence many relevant packages were missing. It was mentioned that the scope of Task View should possibly be reconsidered to avoid overlap with the Phylogenetics Task View.

The maintainer for the **Phlyogenetics** Task View was unresponsive to the Task View editors. In the end it was decided that both the Genetics and Phlyogenetics Task Views should be merged and the current versions should be archived (see issue 17[49] from the issues tab for the full discussion).

In summary the **SocialSciences** and **Multivariate** Task Views were archived as they were too broad. The Task Views covering a wide-ranging subject often do not make it easier for R users to find their required package. These Task Views are also difficult to

[46]Retirement of **SocialSciences** Task View: `https://github.com/cran-task-views/ctv/issues/8`

[47]Proposal of **CausalInference** Task View: `https://github.com/cran-task-views/ctv/issues/15`

[48]Retirement of **Multivariate** Task View: `https://github.com/cran-task-views/ctv/issues/9`

[49]Decision to merge **Phlyogenetics** and **Genetics** Task View: `https://github.com/cran-task-views/ctv/issues/17`

maintain as there many possible packages that could potentially be added to the Task View. The **Genetics** and **Phlyogenetics** Task Views had not been kept up to date, and there was an overlap in the scope of the two task Views.

All four of these Task Views were part of separate clustering pairs derived from the clustering using my similarity measure (see Figure 11) obtained from dependency of packages within Task Views. The reasoning for retirement of these Task Views was mostly due to lack of frequent maintenance or the topics being too broad.

The retirement of these Task Views confirms the selection of similar groups of Task Views selected by my similarity measure. It also highlights the difficulty that is faced by Task View editors of finding maintainers so that Task Views are kept up to date. The discussions also point out that making decisions on which Task Views to create, and how to make sure they still have a sharp focus is not easy. This will become even harder as the CRAN environment continues to grow.

3.3 Enhancing the Similarity Measure with Task View Description Text

3.3.1 Aim

In addition to the package dependency structure, we want to use the text that describes each of the Task Views to enhance the final similarity measure. This text is shown on the Task View web pages, that we referred to in Section 1.4, whose source files where converted to Markdown files. Each piece of text begins with a few paragraphs providing a brief overview of the Task View. Then it lists all of the packages in subtopics alongside a brief description of the separate packages. The text then concludes with a section containing links to related books, papers, blogs. A further description of how the text is formatted and structured is given in the documentation[50] file from the ctv GitHub repository[30].

Using NLP techniques we can measure how similar two pieces of text are, and then combine this similarity measure with the previous similarity measure we have defined (given in equation (4)) using the ratio of the **between** and **within** edges. In the literature there are many cases of using supervised learning to decide on how to weight each similarity in the combined measure. In our case we are performing unsupervised learning, as there is no labeled grouping of Task Views. Therefore I will assign a weighting to each similarity

[50]Documentation for structure and format of Markdown files that render to HTML pages for Task View web pages: `https://github.com/cran-task-views/ctv/blob/main/Documentation.md`

based on what I deem to be more important and what I want my final measure to mean.

3.3.2 Method to Measure Text Similarity

The method we use to measure the similarity of the description files, is to vectorise the documents into numerical feature vectors and then compute the pairwise cosine similarities between them. Where the vectors represent the documents by word occurrences, ignoring the relative position information of the words in the document. This type of vectorisation is known as the "Bag of Words" representation.

Creating the vectors consists of three main steps: tokenizing, counting, and weighting. We follow the process described in the documentation[51] for text-feature extraction of the **scikit-learn** Python (Van Rossum and Drake Jr, 1995) package (Pedregosa et al., 2011).

To weight the occurrence of the words we use the TF-IDF method (the method is discussed further in Robertson, 2004). Different variations of this method are explained in section 6.2.3.4 [52] of the scikit-learn documentation, we define the exact method I use later.

Vectorising the text in this way and taking the cosine similarity is a simple method of computing similarities between texts but has performed well in the past. But this method does not look at the context of the word or their semantic similarity.

The first thing to do is to collect the data, this is straight forward as the Markdown files are all in standardised repositories on the CRAN Task Views GitHub account[29]. We can use the `readLines()` function from the **base** R package (this is a base R package, R Core Team, 2020) in order to extract the text.

The `readLines()` function reads all text lines from a connection (see the documentation[53]), which it takes as an argument. Once we have read the text data from these source files it then needs to be cleaned. This includes removing text that will not be useful in measuring text similarity, such as hyper links and numbers. We also removed package names, as information regarding the package assignment has already been included in the similarity measure using the package dependencies.

[51]**scikit-learn** Python package documentation: `https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction`

[52]Section 6.2.3.4 of scikit-learn documentation: `https://scikit-learn.org/stable/modules/feature_extraction.html#tfidf-term-weighting`

[53]`readLines()` function documentation: `https://stat.ethz.ch/R-manual/R-devel/library/base/html/readLines.html`

We now define the exact vectorisation process we use, once the text has been extracted and cleaned:

1. **Tokenizing**
   Tokenizing is the splitting of a document into tokens, using white spaces and punctuation as token separators. We have chosen tokens to be individual words. Therefore for each document the character string is converted into a character vector containing the separate words. To execute this we use the `unnest_tokens()` function from the **tidytext** R package (Silge and Robinson, 2016).

2. **Lemmatisation**
   The next thing to do is to perform lemmatisation, this the process of converting the inflected forms of a word to the lemma. To implement this we use the `lemmatize_words()` function from the **textstem** R package (Rinker, 2018b). Within this function we use the default dictionary **hash_lemmas** from the **lexicon** R package (Rinker, 2018a) which defines the replacement of each word.

3. **Counting**
   For each of the documents we aggregate the words to get the number of each of the words in the document. We then convert these to frequencies by dividing by the number of unique words in each of the documents (Task Views)

4. **Weighting using TF-IDF**
   From these term frequency vectors we want to increase the weight of words that have higher importance in describing the document. TF-IDF is a method of weighting the vector of tokens.

   TF stands for term frequency, and is the number of times a word appears in the document. We have retrieved this from the previous step.

   IDF stands for inverse document frequency, this term is used to weight the TF of a term. It counts the number of documents where the term appears looking across all the documents that make up the corpus. In our case, the corpus is the description pages of the 36 Task Views (number of Task Views on the 4th of April 2022, see Table 4). IDF gives a larger weighting to words that appear in fewer documents across the corpus. There exists different variations of the IDF term, below we give the formula that we applied.

   IDF of word i with respect to the group of documents D is defined as:

$$\text{IDF}(i, D) = log_2\left(\frac{N}{n_i}\right)$$

where

$$n_i = |\{d \in D : i \in d\}|$$

If a word $i \in d$ appears in every document, $n_i = 36$, the weight $\text{IDF}(i, D)$ will be equal to zero. A word that appears in fewer documents will have a larger weighting. The intuition of this term, is that a term that appears in many documents will not be a good discriminator (Robertson, 2004).

The TF-IDF term for a word i in d with respect to D is

$$\text{TF}(i, d) \times \text{IDF}(i, D)$$

5. **Cosine Similarity**
   Cosine similarity measures the similarity between two vectors by calculating the cosine of the angle between them (see Han, Kamber, and Pei, 2012, Section 2.4.7). Therefore a value close to one implies the vectors are very similar and and a value of minus one means they are very dissimilar. If we normalise the vectors, the cosine between them is equivalent to the dot product. Therefore the cosine similarity between two vectors x and y can be written as:

$$\frac{x^T y}{||x|| \, ||y||}$$

Therefore for two pieces of text that are then vectorised using the method we have given. We define the similarity between the two vectors, which we denote as T_s and T_k, by Equation (5) below.

$$s'(T_s, T_k) = \frac{T_s \cdot T_k}{||T_s|| \, ||T_k||} \tag{5}$$

3.3.3 Combining Similarity Measures

In Section 3.2 we defined a similarity measure between Task Views using the package dependencies (defined by equation (4)). Then in Section 3.3 we gave a method of measuring the similarity between two pieces of text (defined by equation (5)).

We can combine both these similarity measures to create an enhanced similarity measure of Task Views. To combine them we simply add them together and re-weight the contribution of each the separate similarity measures based on how much we want each of the two similarity measures to contribute.

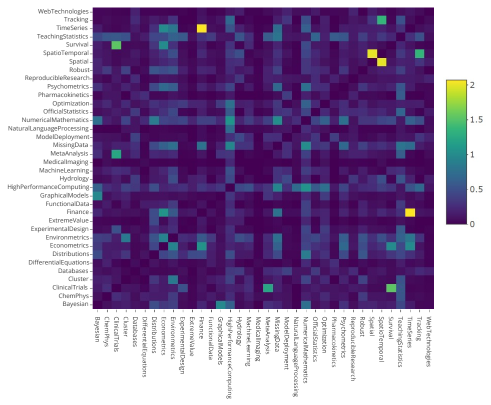

Figure 12: Similarity matrix from similarity measure using package dependencies (s), defined by equation (4).

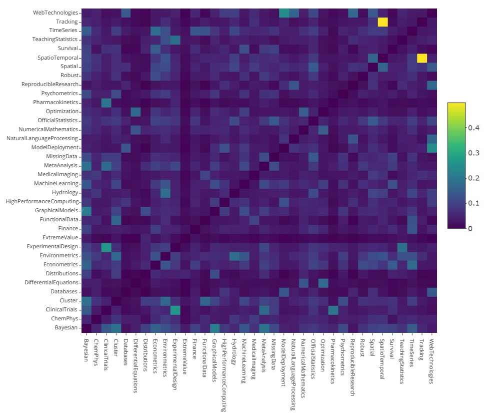

Figure 13: Similarity matrix from similarity measure using text similarity (s'), defined by equation (5).

Figure 12 and Figure 13 are the similarity matrices computed using equation (4) and equation (5) respectively applied to the snapshot taken on the 4th of April 2022.

The final similarity measure between two Task Views, i and j, will therefore be defined by equation (6).

$$S(i,j) \;=\; w_1 \cdot s(V_i,V_j) \;+\; w_2 \cdot s'(T_i,T_j); \qquad w_1 + w_2 = 1 \tag{6}$$

Choosing the value of the weights w_1 and w_2 lets us decide how how much influence that each variable has in determining the overall similarity. To select these weights we look at the relative contribution to the average object similarity measure over all pairs of observations. This method is described in section 14.3.3 of Hastie, Tibshirani, and J. H. Friedman (2009), they used the method to combine dissimilarity measures.

We can rewrite the average of the combined similarity matrix over all observations in terms of the averages of the two separate similarity matrices.

$$\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} S(x_i, x_j) \;=\; w_1 \cdot \bar{s} \;+\; w_2 \cdot \bar{s'}$$

with

$$\bar{s} = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} s(x_i, x_j) \qquad \text{and} \qquad \bar{s'} = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} s'(x_i, x_j)$$

This shows the influence of the similarity component s on the combined average is $w_1 \cdot \bar{s}$ and $w_2 \cdot \bar{s'}$ for the component s'.

The similarity measure based on package dependencies (s), identified Task Views as being close that were then consequently removed by the Task View maintainers. We therefore want the majority of the information to come from this component.

Hence we set the weights so that 60% of the total average is explained by the package dependencies similarity. So w_1 is given by equation (7) below.

$$w_1 = \frac{(0.6) \cdot \bar{s'}}{(0.4) \cdot \bar{s} \;+\; (0.6) \cdot \bar{s'}} \tag{7}$$

4 Task View Recommendations

4.1 Aim

The Task Views are maintained by volunteers and are always welcome to contributions for additional content from members of the community. Due to the huge number of packages that are in CRAN it is infeasible for a Task View maintainer to review all of the available packages. Therefore, a model that would give a small selection of high quality suggestions for the maintainer to then review would be useful.

Such a model would also be useful from the perspective of a package developer. After publishing their package onto CRAN, they could then check whether they should propose their package to one of the Task View maintainers.

The similarity measure that we created in the previous section, is constructed in a way such that it cannot be used to measure the similarity of a single package to a Task View. This being the case, we need to construct a new model to give suggestions.

The model we want to construct should give a Task View suggestion after being giving some set of features of an unassigned package. However, it would not be reasonable to assign a Task View to every package. This would depart from the objective of Task Views, which is to give a sharp focus on packages that are needed for a task. For this reason, the model should also have the possibility of assigning a package to no Task View.

We train the model using the CRAN snapshot taken on the 4th of April 2022 (Table 4 in the Appendix gives a summary of the snapshot)). Consequently, the model should have a multicategory response variable, made up of 37 categories. The 36 Task Views with an additional "None" category.

4.2 Multinomial Logistic Regression

To handle this multi-class classification problem we use a multinomial logistic model. This is a generalization of logistic regression for multinomial response variables. Once the model is fitted, when given a set of predictors it will output a probability vector with length of the number of possible classes.

To describe the logistic regression model we let y_i denote the outcome for observation i, and $\mathbf{x}_i$ the p-dimensional vector of predictors for observation i.

In logistic regression, a response for an observation i is seen as an independent trial from a Bernoulli distribution with probability π_i (y_i takes values in $\{0, 1\}$). The probability of observation i is therefore given by equation (8).

$$\mathbb{P}(y_i|\pi_i) = \pi_i^{y_i}(1-\pi_i)^{1-y_i} \tag{8}$$

The logistic regression model is constructed by using the the sigmoid function, $\sigma(x) := \frac{\exp(x)}{1+\exp(x)}$. Then introducing the parameter vector $\boldsymbol{\beta}$ and setting $\sigma(\mathbf{x}_i^T\boldsymbol{\beta}) = \pi_i$. Therefore the class probabilities are described as follows:

$$\begin{aligned} \mathbb{P}(y_i = 1|\mathbf{x}_i) &= \frac{1}{1+\exp(-\mathbf{x}_i^T\boldsymbol{\beta})} \\ \mathbb{P}(y_i = 0|\mathbf{x}_i) &= \frac{1}{1+\exp(\mathbf{x}_i^T\boldsymbol{\beta})} \end{aligned} \tag{9}$$

Which implies that:

$$\log\left(\frac{\mathbb{P}(y_i = 1|\mathbf{x}_i)}{\mathbb{P}(y_i = 0|\mathbf{x}_i)}\right) = \mathbf{x}_i^T\boldsymbol{\beta} \tag{10}$$

see Hastie, Tibshirani, and J. H. Friedman 2009, Section 4.4 for more details on logistic regression models.

For multinomial logistic regression, we allow for more than two categories and we model the response of an observation as the result of a single multinomial trial. We denote this outcome by $\mathbf{y}_i = (y_{i1}, \dots, y_{ic})$, where $y_{ij} = 1$ when the response is in category j and $y_{ij} = 0$ otherwise. Where c denotes the number of possible categories, which in our case is 37.

We let π_{ij} denote the probability of outcome y_{ij} occurring. In our case π_{ij} is the probability of allocating the package i with predictors $\mathbf{x}_i$ to Task View j, where $\mathbf{y}_i$ represents a single Task View allocation for the predictors $\mathbf{x}_i$. The probability mass function for the multinomial distribution for outcome i is given by:

$$\mathbb{P}(\mathbf{y}_i|\boldsymbol{\pi}_i) = \prod_{j=1}^{37} \pi_{ij}^{y_{ij}}$$

with:

$$\sum_{j=1}^{c} y_{ij} = 1 \quad \text{and} \quad \sum_{j=1}^{c} \pi_{ij} = 1$$

Multinomial logistic models simultaneously describe the log odds for all $\binom{c}{2}$ possible pairs of categories. To parameterise this, the traditional approach is the baseline-category logit model, where equation (10) from logistic regression is extended to $c-1$ logits. Where each of the response categories is paired with a baseline category, and the jth baseline-category logit is defined as $\log\left(\frac{\pi_{ij}}{\pi_{ic}}\right)$.

$c-1$ parameter vectors $\boldsymbol{\beta}_j$ are introduced, and the baseline-category logit model is defined by these equations:

$$\log\left(\frac{\pi_{ij}}{\pi_{ic}}\right) = \mathbf{x}_i^T \boldsymbol{\beta}_j \ , \quad j = 1, \ldots, c-1 \tag{11}$$

Therefore a separate parameter vector is introduced for each baseline-category logit. Further details of this model is described in Agresti (2015, Section 6.1).

Another parameterisation is the one taken by J. Friedman, Hastie, and Tibshirani (2010). Here we have c parameter vectors, and the model is given by:

$$\pi_{ij} = \frac{\exp(\mathbf{x}_i^T \boldsymbol{\beta}_j)}{\sum_{k=1}^{c} \exp(\mathbf{x}_i^T \boldsymbol{\beta}_k)} \ , \quad j = 1, \ldots, c \tag{12}$$

This is a more symmetric approach, however if we add any constant vector to the parameter vectors we retrieve identical probabilities. Therefore, the value of the parameters cannot be estimated without the introduction of constraints. This parameterisation was used in the context of regularisation which then handles this issue.

4.3 Model Fitting

4.3.1 Data

When deciding on a set of packages to train and test the model, it would not make sense to include all packages that are not assigned a Task View. The large proportion of these packages would have not been reviewed by Task View maintainers, and so would not be representative for packages belonging to the “None” category.

Therefore to choose the packages to include that are labelled as belonging to the “None” category, we select packages that have a high amount of monthly downloads. This chosen threshold of monthly downloads will act as a proxy for selecting a set of packages that will be labeled as not being assigned to a Task View. In order to decide on this threshold we use the **cranlogs** R package (Csárdi 2019, the package is described in Section 2.2.2 of this report).

Within the Task Views the top 25% downloaded packages have more than 2500 monthly downloads. We use packages with no assigned Task Views that have more monthly downloads than this threshold as a proxy for packages that have been rejected by Task View maintainers.

In total there are 3514 packages that have at least one assigned Task View, and 1245 not assigned packages that meet the decided monthly download threshold. Combining these gives us a total number of 4759 observations, we then split this data set into a training and testing set with an 80:20 ratio for each set respectively.

In addition, we have the rest of the packages that have no Task View assignment that do not meet the monthly download threshold (14259 packages). After fitting the model, we make predictions with the fitted model on this set of packages to generate suggestions for packages to be added to Task Views.

An important point is how packages with multiple Task View allocations are handled. We have represented these packages as separate observations that have the same set of predictors but different response labels.

4.3.2 Features

Throughout the report we covered two main types of data related to the packages. The dependency structure of packages and authors, and then the surrounding text meta-data of the packages. We incorporate both types of data to define features for the model, to predict allocation to a Task View.

In Section 3.3 we already carried out the text retrieval, cleaning and created vectorized versions of each of the Task View sources. Additionally to this, on CRAN for each package there exists an extended title and a further piece of text giving an overview of the packages functionalities. This short description of the package can be seen under the title of the respective package web page[11]. Therefore we can calculate the similarity of each of the Task View texts with the text data from the individual packages.

To execute this we combine the title and description texts of each package, and we then vectorize the text in a similar way as described in Section 3.3.2. We tokenize and lemmatize as described previously, and weight the words using the TF-IDF weights computed from the Task View sources. We have not recalculated the IDF weights using the individual package text, therefore words that appear in the package text but do not appear in the Task View sources will not be taken into account. Then we compute the cosine similarity of the vectorized package text to each of the vectorized Task View text to return 36 features.

We also include the dependency structure of the packages as features of the model. We look at the immediate hard dependencies of packages and then calculate the proportion of the assignment of these packages to Task Views and no assignment. This creates 37 features.

Moreover we have computed the proportion of Task Views that that the authors of each package worked on. This creates another 37 features.

4.3.3 Model Tuning

To tune the model we use LASSO in order to create a classifier with a good prediction accuracy.

LASSO solves the problem given by equation (13).

$$\min_{\beta} \frac{1}{N} \sum_{i=1}^{N} l(y_i, \beta^T x_i) \; + \; \lambda \|\beta\|_1 \tag{13}$$

Where $l(y_i, \eta_i)$ is the negative log-likelihood contribution for observation i. LASSO can be carried out in R using the **glmnet** R package (J. Friedman, Hastie, and Tibshirani, 2010). There is a vignette[54] with this package that provides an extensive description

[54]Vignette for the **glmnet** package: https://cran.r-project.org/web/packages/glmnet/vignettes/glmnet.pdf

of the different models you can fit. This package was made publicly available alongside the paper by J. Friedman, Hastie, and Tibshirani (2010). For the case of multinomial regression they used the parameterisation of the class probabilities given by equations (12).

The model is fit under the multinomial likelihood and therefore the negative log-likelihood for N independent observations is given by equation (14).

$$- l(y_i, \pi_{ij}) \;=\; -\sum_{j=1}^{37} y_{ij} \log(\pi_{ij}) \tag{14}$$

Now replacing the class probabilities, π_{ij}, with the parameterisation given by equations (12), the LASSO penalized negative log-likelihood function (13) is then given by equation (15).

$$\min_{\beta} \frac{1}{N} \sum_{i=1}^{N} \Big[\sum_{j=1}^{37} y_{ij}(\mathbf{x}_i^T \boldsymbol{\beta}_j) - \log(\sum_{j=1}^{37} \exp(\mathbf{x}_i^T \boldsymbol{\beta}_j)) \Big] \;+\; \lambda \|\beta\|_1 \tag{15}$$

The `glmnet()` function solves the problem given in equation (15) over a grid of λ values (page 2 of vignette[54]). This is a decreasing sequence of λ's starting from the the smallest value λ_{max} that causes $\hat{\boldsymbol{\beta}}_j$ to be equal to zero $\forall j \in \{1, \ldots, c\}$ (see J. Friedman, Hastie, and Tibshirani, 2010, section 2.5 for full details). This creates a model for each λ value in the grid of λ values.

In order to choose a λ a common method is to use cross-validation, and we can perform this using the `cv.glmnet()` function. For each λ this now performs cross-validation, giving as a performance measure. The data is split into ten folds (the default) and for each fold the model is fit on the rest of the data then the performance of the model is calculated on the fold.

The default loss function used to calculate the performance for multinomial regression is the multinomial deviance. The deviance for multinomial models is two times the difference of the log-likelihood of the saturated model minus the log-likelihood of the model we are fitting (see Agresti, 2015, Section 6.1.4).

For the multinomial case the likelihood for the saturated model is just one, therefore the multinomial deviance is equal to minus two times the log-likelihood of the fitted model (this is also true for the binomial model, see J. Friedman, Hastie, and Tibshirani, 2010, Caption of Figure 2 in Section 6).

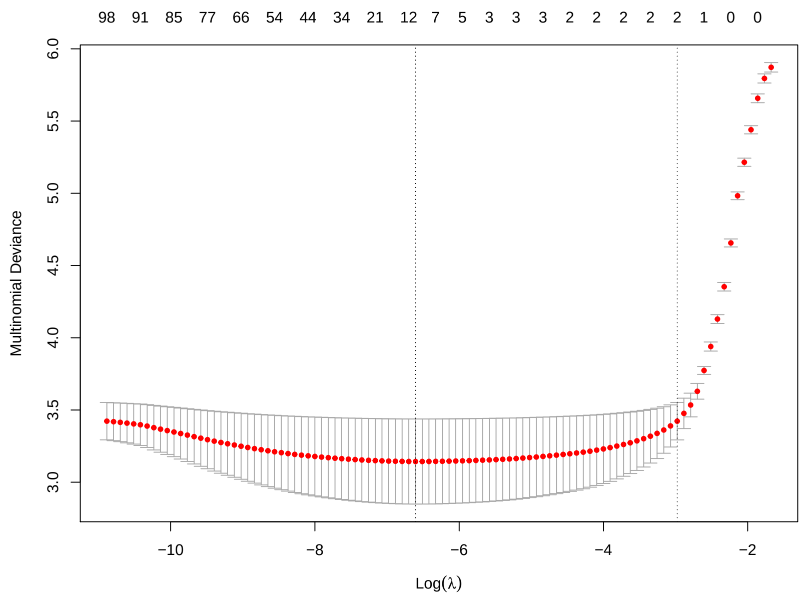

Figure 14: This figure shows the multinomial deviance against the different λ values. The red points are the average values of the multinomial deviance across the folds. The vertical bars on each point are the standard deviations of these values (standard error). The x coordinate of the left vertical line is the λ value that gives the minimum average value (`lambda.min`). The other vertical line indicates the λ value that gives the average error within one standard deviation of the minimum (`lambda.1se`).

4.3.4 Model Accuracy and Recommendations

After running `cv.glmnet()` we select the λ value that gave the lowest average multinomial deviance across the folds (the λ value indicated by the left vertical line in Figure 14). This λ value is denoted as `lambda.min` in the documentation of the **glmnet** R package. For this chosen value of λ we then test the performance of the model on the data that we held out. Using the largest predicted probabilities as the predicted class and then taking the proportion of correct classifications to the number of observations in the test set we get an accuracy of 80.44%.

In Section 4.3.1 we described the data we use to train and test the model, and mentioned that we have 14259 packages that are not assigned a Task View and do not meet the monthly download threshold we set. Now that the model is fit, we can look at the predictions of the model on this set of packages. This generates recommendations for the existing Task Views.

For each of the Task Views we look at the packages that have the highest predicted probabilities. We give the top five packages with highest probability in Table 8 in the Appendix.

4.3.5 Feedback from Maintainers

From the model predictions given in Table 8 I emailed four Task View maintainers, with suggestions for packages to add to their respective Task Views. I only sent a small selection of packages as recommendations.

I suggested to the NaturalLanguageProcessing Task View[55] maintainer (Fridolin Wild) to add the **word2vec** package, he agreed and I have therefore sent a pull request[56] via GitHub.

In addition, I emailed the Psychometrics Task View[57] maintainer (Patrick Mair). I suggested the **Rirt** and **dextergui** package, and he agreed that both should belong to the Task View. The addition of these packages to the Task View can be seen via the commit[58] on the Markdown file hosted on the Psychometrics Task View GitHub repository.

I also emailed the Environmetrics Task View[59] maintainer (Gavin Simpson) with package suggestions, he replied with "By the looks of some of those, they should be added with other packages and may need their own section(s)." He asked for me to send a pull request that makes the relevant changes to the Task View that includes a summary sentence as to what the package does. I ran out of time to do this, but this emphasises the importance of domain knowledge for Task View maintenance.

[55]NaturalLanguageProcessing Task View web page: `https://cran.r-project.org/web/views/NaturalLanguageProcessing.html`

[56]Pull request to add **word2vec** R package to the NaturalLanguageProcessing Task View (waiting to be accepted as of the 4th of May 2022): `https://github.com/cran-task-views/NaturalLanguageProcessing/pull/3`

[57]Psychometrics Task View web page: `https://cran.r-project.org/web/views/Psychometrics.html`

[58]Commit on the Psychometrics Task View Markdown showing the addition of the suggested packages: `https://github.com/cran-task-views/Psychometrics/commit/481f17b0e4a3184823806d8c7971f2e397b406b1`

[59]Environmetrics Task View web page: `https://cran.r-project.org/web/views/Environmetrics.html`

5 Discussion and Conclusion

5.1 Similarity of Task Views

In Section 3 we constructed a similarity measure $S(i, j)$ that measures the similarity of any two Task Views i and j. This was constructed as a weighted combination of two similarity measures that each utilized different attributes of the Task Views.

We first constructed the similarity measure $s(V_i, V_j)$, that looked at the number of hard package dependencies travelling from one Task View to another. We then looked at utilizing the text data to help measure similarity between Task View.

Extracting the text that is displayed on the Task View web pages before the transfer to GitHub had been completed, would have been much more laborious. Therefore for the CRAN snapshot taken on the 20th of February 2022, we just looked at the results of the similarity measure $s(V_i, V_j)$.

We found that the pairs chosen by clustering with this similarity measure coincided with the four Task Views that were decided to be archived by the Task View editors.

Once the transfer was completed we could extract the text data, and use this to enhance the similarity measure. The process of vectorizing the Task View text then became useful when creating features for the Task View recommendation model in Section 4.

After the official announcement of the new Task View workflow was completed, the Task View editors started to consider new Task view proposals[33,47,60]. In the documentation[61] for proposing a new Task View there is an emphasis on avoiding overlap with other Task Views. Therefore our similarity measure could be useful to see which Task Views the newly proposed Task View is similar to. This may help proposers and reviewers decide if the scope of new Task View should be sharpened or avoided completely.

In conclusion, the similarity measure we constructed using package dependencies agreed with retirement decisions made by the Task View editors. We can not just look at the overlap of packages between Task Views, as for example there exist many packages in CRAN that have the same aims. In addition some packages can have functionalities that can motivate them belonging to multiple Task Views. We need to look at whether the scopes of the Task Views are similar, we tackled this by using package dependencies.

[60]Proposal of **InfectiousEpi** Task View: `https://github.com/cran-task-views/ctv/issues/25`

[61]Documentation for proposing a new Task View: `https://github.com/cran-task-views/ctv/blob/main/Proposal.md`

5.2 Task View Recommendations

In Section 4 we used a multinomial logistic regression model to classify packages to a Task View. We dealt with multiple assigned packages by having separate observations with identical features but with different labels. This departs from the assumptions made in multinomial regression, where we assume that we have N independent observations from single multinomial trials. If a multi-labeled package is assigned to a Task View we then know that the next trial will not be the same Task View again.

The multiple assignment in the data suggests that using a multi-label classifier would be a more sound approach. In such a model, it would be possible for a package to have a large predicted probability for multiple Task Views. However, we used a multi-class model that outputs a probability vector and therefore will avoid multi-label suggestions. As the aim of the Task View editors is to create Task Views with a sharp focus and to avoid overlap, it could be argued that our approach is more suitable.

Our model had an accuracy of 80% on a hold out data set, and using this model we created a table of recommendations given in Table 8 in the Appendix. If we review the packages in Table 8 and compare these suggestions to the respective package descriptions, the majority of these recommendations seem reasonable. Of course to check the quality of these recommendations we should provide them to further Task View maintainers and ask for their feedback.

In conclusion, using: package dependencies, author collaborations and the text information we created a model with a good classification accuracy on a hold out set. However, in order to determine the quality of the predictions we would need more feedback from the Task View maintainers. We would then be able to decide if this model can be used in practice to provide recommendations to the maintainers.

6 Bibliography

Articles

Antiqueira, Lucas and Luciano da Fontoura Costa (Jan. 2009). “Characterization of subgraph relationships and distribution in complex networks”. In: *New Journal of Physics* 11.1, p. 13058. DOI: 10.1088/1367-2630/11/1/013058.

Bella, Enrico di, Luca Gandullia, and Sara Preti (2021). “Analysis of scientific collaboration network of Italian Institute of Technology”. In: *Scientometrics* 126.10, pp. 8517–8539. ISSN: 1588-2861. DOI: 10.1007/s11192-021-04120-9.

Freeman, Linton C (1977). “A Set of Measures of Centrality Based on Betweenness”. In: *Sociometry* 40.1, pp. 35–41. DOI: 10.2307/3033543.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani (2010). “Regularization Paths for Generalized Linear Models via Coordinate Descent”. In: *Journal of Statistical Software* 33.1, pp. 1–22. URL: https://www.jstatsoft.org/v33/i01/.

Grolemund, Garrett and Hadley Wickham (2011). “Dates and Times Made Easy with lubridate”. In: *Journal of Statistical Software* 40.3, pp. 1–25. URL: https://www.jstatsoft.org/v40/i03/.

Joo, Rocío et al. (2020). “Navigating through the r packages for movement”. In: *Journal of Animal Ecology* 89.1, pp. 248–267. DOI: https://doi.org/10.1111/1365-2656.13116.

Newman, M E J (June 2001a). “Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality”. In: *Physical Review E* 64.1, p. 16132. DOI: 10.1103/PhysRevE.64.016132.

— (Jan. 2001b). “The structure of scientific collaboration networks”. In: *Proceedings of the National Academy of Sciences* 98.2, p. 404. DOI: 10.1073/pnas.98.2.404.

Pedregosa, Fabian et al. (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12.85, pp. 2825–2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

Robertson, Stephen (Jan. 2004). “Understanding inverse document frequency: on theoretical arguments for IDF”. In: *Journal of Documentation* 60.5, pp. 503–520. DOI: 10.1108/00220410410560582.

Shahin Tavakoli (2017). “Multivariate Statistics (ST323 ST412)”. In: URL: https://stavakoli.com/activities/multivariate_warwick/.

Books

Agresti, Alan (2015). *Foundations of linear and generalized linear models.* Wiley series in probability and statistics. Hoboken, New Jersey: John Wiley & Sons Inc. ISBN: 978-1-118-73003-4.

Han, Jiawei, Micheline Kamber, and Jian Pei (Jan. 2012). *2 - Getting to Know Your Data.* en. The Morgan Kaufmann Series in Data Management Systems. Boston: Morgan Kaufmann, pp. 39–82. DOI: 10.1016/B978-0-12-381479-1.00002-2.

Hastie, Trevor, Robert Tibshirani, and J H Friedman (2009). *The elements of statistical learning: data mining, inference, and prediction.* 2nd ed. Springer series in statistics. New York, NY: Springer. URL: https://hastie.su.domains/ElemStatLearn/.

Kolaczyk, Eric D and Gábor Csárdi (2014). *Statistical Analysis of Network Data with R.* Vol. 65. Use R! New York, NY: Springer New York. DOI: 10.1007/978-1-4939-0983-4.

Newman, Mark (2018). *Networks.* eng. 2nd ed. Oxford: Oxford University Press. DOI: 10.1093/oso/9780198805090.001.0001.

Van Rossum, Guido and Fred L Drake Jr (1995). *Python tutorial.* Amsterdam, The Netherlands: Centrum voor Wiskunde en Informatica. ISBN: 978-1-4414-1269-0.

Wickham, Hadley and Jennifer Bryan (2015). *R Packages.* en. 2nd. O’Reilly Media. URL: https://r-pkgs.org/.

R Packages and Documentation

Allaire, J.J. et al. (2017). *networkD3: D3 JavaScript Network Graphs from R.* R package version 0.4. URL: https://CRAN.R-project.org/package=networkD3.

Almende B.V. et al. (2021). *visNetwork: Network Visualization using 'vis.js' Library*. URL: https://CRAN.R-project.org/package=visNetwork.

Csardi, Gabor and Tamas Nepusz (2006). *The igraph software package for complex network research*. URL: https://igraph.org.

Csárdi, Gábor (2019). *cranlogs: Download Logs from the 'RStudio' 'CRAN' Mirror*. URL: https://CRAN.R-project.org/package=cranlogs.

Kiener, Patrice (2021). *RWsearch*. URL: https://CRAN.R-project.org/package=RWsearch.

Kosmidis, Ioannis (2019). *cranly: Package Directives and Collaboration Networks in CRAN*. URL: https://CRAN.R-project.org/package=cranly.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. Vienna, Austria. URL: https://www.R-project.org/.

Rinker, Tyler W (2018a). *lexicon: Lexicon Data*. Buffalo, New York. URL: http://github.com/trinker/lexicon.

— (2018b). *textstem: Tools for stemming and lemmatizing text*. Buffalo, New York. URL: http://github.com/trinker/textstem.

Silge, Julia and David Robinson (2016). *tidytext: Text Mining and Analysis Using Tidy Data Principles in R*. DOI: 10.21105/joss.00037.

7 Appendix

Packages	Authors	Task Views	Packages assigned to Task Views	Median packages per author	Median author per package
18966	27873	40	3423	1	2

Table 3: Summary of CRAN (20th February 2022)

Packages	Authors	Task Views	Packages assigned to Task Views	Median packages per author	Median author per package
19018	28023	36	3514	1	2

Table 4: Summary of CRAN (4th April 2022)

Task Views	20th Feb 2022		4th April 2022	
	Packages	Authors	Packages	Authors
Bayesian	138	385	223	624
ChemPhys	81	174	79	165
ClinicalTrials	59	142	59	144
Cluster	109	310	108	309
Databases	37	85	42	94
DifferentialEquations	28	116	28	116
Distributions	249	565	266	614
Econometrics	140	348	144	358
Environmetrics	91	363	91	365
ExperimentalDesign	107	201	115	216
ExtremeValue	24	56	39	92
Finance	166	424	166	429
FunctionalData	42	121	43	123
Genetics	21	125	-	-
GraphicalModels	32	108	31	104
HighPerformanceComputing	87	307	84	305
Hydrology	101	253	101	256
MachineLearning	101	488	102	493
MedicalImaging	23	50	33	54
MetaAnalysis	152	337	160	359
MissingData	165	584	213	737
ModelDeployment	32	160	32	149
Multivariate	110	362	-	-
NaturalLanguageProcessing	54	111	54	112
NumericalMathematics	115	267	115	268
OfficialStatistics	131	355	131	319
Optimization	130	306	135	321
Pharmacokinetics	18	51	30	106
Phylogenetics	81	276	-	-
Psychometrics	239	545	232	535
ReproducibleResearch	100	446	102	518
Robust	58	140	60	143
SocialSciences	77	252	-	-
Spatial	174	544	198	608
SpatioTemporal	85	293	88	299
Survival	247	576	248	580
TeachingStatistics	44	192	45	197
TimeSeries	342	786	344	798
Tracking	45	134	51	158
WebTechnologies	199	403	203	429

Table 5: Number of packages and authors in each of the snapshots. The Task Views in red are those that were archived during the transfer process of the CRAN Task View initiative from R-Forge to GitHub.

Task Views	Proportion of Largest Component(%)	Average Distance
Bayesian	47.9	3.8
ChemPhys	66.1	3.7
ClinicalTrials	27.8	2.3
Cluster	56.3	3.5
Databases	89.4	2.4
DifferentialEquations	87.9	2.8
Distributions	62.5	3.6
Econometrics	72.9	2.8
Environmetrics	78.4	3.1
ExperimentalDesign	10.6	3.5
ExtremeValue	53.3	3.0
Finance	69.7	2.8
FunctionalData	22.8	1.8
GraphicalModels	67.3	2.5
HighPerformanceComputing	88.2	3.0
Hydrology	37.1	3.5
MachineLearning	86.6	3.1
MedicalImaging	51.9	2.1
MetaAnalysis	27.3	3.8
MissingData	64.6	3.5
ModelDeployment	95.3	2.7
NaturalLanguageProcessing	90.2	3.6
NumericalMathematics	82.8	2.9
OfficialStatistics	59.2	3.8
Optimization	68.5	3.4
Pharmacokinetics	59.4	2.2
Psychometrics	61.9	3.3
ReproducibleResearch	91.1	2.8
Robust	69.9	2.6
Spatial	82.4	3.2
SpatioTemporal	69.9	2.7
Survival	50.5	3.8
TeachingStatistics	93.4	2.7
TimeSeries	61.5	3.9
Tracking	22.2	3.1
WebTechnologies	83.0	3.0

Table 6: Size of the largest component as a proportion of the total number of authors in each subnetwork. And the average geodesic distance between authors.

Task Views	Top 5 Influential Authors				
Bayesian	R Core (0.09)	Achim Zeileis (0.04)	Roger Bivand (0.04)	Rob Hyndman (0.04)	Ben Goodrich (0.03)
ChemPhys	R Core (0.19)	Ben Bolker (0.15)	RStudio (0.11)	Uwe Ligges (0.1)	Claudia Beleites (0.06)
ClinicalTrials	Torsten Hothorn (0.02)	Detlew Labes (0.01)	Thomas Lumley (0.01)	Achim Zeileis (0.01)	Bill Venables (0.01)
Cluster	Kurt Hornik (0.09)	Martin Maechler (0.09)	R Core (0.09)	Pavel N Krivitsky (0.05)	Friedrich Leisch (0.05)
Databases	RStudio (0.21)	Yuan Tang (0.13)	Jeroen Ooms (0.12)	Hadley Wickham (0.11)	Austin Dickey (0.11)
DifferentialEquations	Hadley Wickham (0.17)	Simon N Wood (0.16)	Steven G Johnson (0.15)	Dirk Eddelbuettel (0.12)	Matteo Fasiolo (0.11)
Distributions	R Core (0.09)	Martin Maechler (0.07)	Ben Bolker (0.06)	Christophe Dutang (0.04)	Achim Zeileis (0.04)
Econometrics	Achim Zeileis (0.12)	R Core (0.07)	Roger Bivand (0.05)	Ben Bolker (0.04)	Matthieu Stigler (0.04)
Environmetrics	Ben Bolker (0.15)	R Core (0.14)	Jeff Laake (0.1)	Martin Maechler (0.07)	Kurt Hornik (0.05)
ExperimentalDesign	Yang Li (0.01)	Ritsert Jansen (0)	Jack Dongarra (0)	Bill Venables (0)	Ulrike Groemping (0)
ExtremeValue	Martin Maechler (0.11)	Alec Stephenson (0.08)	R Core (0.05)	Alan Genz (0.05)	Simone Padoan (0.04)
Finance	Martin Maechler (0.1)	Dirk Eddelbuettel (0.08)	Christophe Dutang (0.07)	R Core (0.05)	Joshua Ulrich (0.04)
FunctionalData	Luo Xiao (0.01)	Sonja Greven (0.01)	Giles Hooker (0.01)	Jona Cederbaum (0.01)	Ciprian Crainiceanu (0)
GraphicalModels	Ben Bolker (0.15)	Martin Maechler (0.15)	Pavel N Krivitsky (0.06)	Hadley Wickham (0.06)	Douglas Bates (0.05)
HighPerformanceComputing	R Core (0.19)	Yuan Tang (0.13)	Dirk Eddelbuettel (0.09)	Kristian Hovde Liland (0.08)	Paul Hiemstra (0.07)
Hydrology	Edzer Pebesma (0.05)	Scott Chamberlain (0.05)	Sam Albers (0.02)	David Blodgett (0.02)	Joseph Stachelek (0.02)
MachineLearning	Dirk Eddelbuettel (0.38)	Christoph Bergmeir (0.1)	Yuan Tang (0.08)	R Core (0.07)	Martin Maechler (0.06)
MedicalImaging	Jon Clayden (0.18)	Karsten Tabelow (0.05)	John Muschelli (0.04)	Brandon Whitcher (0.02)	Chris Rorden (0)
MetaAnalysis	Ben Bolker (0.03)	R Core (0.02)	Thomas Lumley (0.01)	Brenton M Wiernik (0.01)	Malcolm Barrett (0.01)
MissingData	Dirk Eddelbuettel (0.07)	Ben Goodrich (0.05)	R Core (0.04)	Martin Maechler (0.04)	Ben Bolker (0.03)
ModelDeployment	Yuan Tang (0.31)	Kurt Hornik (0.19)	RStudio (0.17)	Michael Hahsler (0.12)	Microsoft Corporation (0.08)
NaturalLanguageProcessing	Kurt Hornik (0.23)	Jan Wijffels (0.22)	Brian Ripley (0.19)	Kenneth Benoit (0.18)	Johannes Gruber (0.18)
NumericalMathematics	Martin Maechler (0.11)	Kurt Hornik (0.11)	Timothy A Davis (0.1)	R Core (0.09)	Ravi Varadhan (0.06)
OfficialStatistics	R Core (0.15)	Matthias Templ (0.05)	Achim Zeileis (0.05)	Brian Ripley (0.05)	Dirk Eddelbuettel (0.04)
Optimization	Dirk Eddelbuettel (0.11)	Kurt Hornik (0.1)	Martin Maechler (0.07)	Olaf Mersmann (0.06)	Brian Ripley (0.05)
Pharmacokinetics	Bill Denney (0.09)	Justin Wilkins (0.07)	Mats O Karlsson (0.06)	Matthew Fidler (0.05)	Dirk Eddelbuettel (0.02)
Psychometrics	Ben Goodrich (0.07)	Achim Zeileis (0.05)	R Core (0.04)	Kurt Hornik (0.04)	Alexander Robitzsch (0.04)
ReproducibleResearch	Noam Ross (0.13)	Pierre Formont (0.11)	Yihui Xie (0.09)	Nan Xiao (0.08)	Hadley Wickham (0.07)
Robust	Martin Maechler (0.21)	Uwe Ligges (0.05)	R Core (0.05)	Matias Salibian-Barrera (0.05)	Kurt Hornik (0.04)
Spatial	Roger Bivand (0.18)	R Core (0.07)	Hadley Wickham (0.07)	Edzer Pebesma (0.05)	Brian Ripley (0.05)
SpatioTemporal	R Core (0.09)	Xianghui Dong (0.08)	Dirk Eddelbuettel (0.08)	Michael Sumner (0.06)	Roger Bivand (0.05)
Survival	Martin Maechler (0.05)	R Core (0.05)	Ben Bolker (0.04)	Thomas Lumley (0.03)	Mark Stevenson (0.02)
TeachingStatistics	Hadley Wickham (0.16)	R Core (0.12)	Randall Pruim (0.1)	Achim Zeileis (0.09)	Ben Baumer (0.07)
TimeSeries	Rob Hyndman (0.05)	Kurt Hornik (0.05)	Martin Maechler (0.05)	R Core (0.05)	Toby Hocking (0.04)
Tracking	Michael Sumner (0.02)	Josh OBrien (0.02)	Edzer Pebesma (0.02)	Daniel Herszenhut (0.01)	Ian Jonsen (0.01)
WebTechnologies	Hadley Wickham (0.1)	Bob Rudis (0.1)	Scott Chamberlain (0.06)	Chester Ismay (0.05)	RStudio (0.05)

Table 7: Top five authors with largest normalised betweenness values, in decreasing order from left to right

Task Views	Top 5 Task View recommendations				
Bayesian	bayest	BayesRS	qbld	stableGR	bmabasket
ChemPhys	readMzXmlData	readJDX	readBrukerFlexData	SpecHelpers	MALDIquantForeign
ClinicalTrials	MCPModPack	MedianaDesigner	VEwaningVariant	VEwaning	CorrMixed
Cluster	MatManlyMix	MatTransMix	clustMD	ManlyMix	netClust
Databases	rollbar	taxizedb	jetpack	cranlogs	tidyquery
DifferentialEquations	bvpSolve	TestGardener	rTorch	PSPManalysis	deTestSet
Distributions	MPkn	orders	inferr	ollggamma	good
Econometrics	frmpd	JFE	GVARX	iClick	iForecast
Environmetrics	openCR	secrlinear	secrdesign	rWind	Rdistance
ExperimentalDesign	gen2stage	ph2mult	factoptd	ADCT	Opt5PL
ExtremeValue	clinUtils	patientProfilesVis	clinDataReview	inTextSummaryTable	distillery
Finance	monotonicity	HierPortfolios	portfolio	PortfolioEffectEstim	mfGARCH
FunctionalData	refund.shiny	fdaPOIFD	conformalInference.fd	funcharts	rofanova
GraphicalModels	GGMselect	tergmLite	dnr	pchc	classGraph
HighPerformanceComputing	RViennaCL	clustra	Rrdrand	pbatR	apTreeshape
Hydrology	gumboot	CSHShydRology	countyfloods	dbhydroR	hydroToolkit
MachineLearning	InvariantCausalPrediction	cornet	palasso	tmle	partitionMap
MedicalImaging	fmriqa	cmR	nucim	loder	wal
MetaAnalysis	simdistr	CKAT	GMMAT	attenuation	GWASinspector
MissingData	mifa	imputeGeneric	doMIsaul	linkim	miceafter
ModelDeployment	cumulocityr	plumberDeploy	neighbr	sparktf	rHealthDataGov
NaturalLanguageProcessing	word2vec	LSX	text.alignment	doc2vec	R.temis
NumericalMathematics	jordan	weyl	EigenR	go2bigq	Rfractran
OfficialStatistics	samplingEstimates	svydiags	svrep	SamplingBigData	mase
Optimization	lpmodeler	ROI.plugin.alabama	ROI.plugin.quadprog	ROI.models.netlib	ROI.plugin.nloptr
Pharmacokinetics	PKPDsim	mapbayr	AMGET	pmxpartab	xpose4
Psychometrics	mstR	dextergui	Rirt	eirm	deltaPlotR
ReproducibleResearch	getable	ezknitr	metathis	rrtable	listdown
Robust	pyinit	RMBC	RSC	DetMCD	rrcov3way
Spatial	tilegramsR	GISTools	rasterpic	SpatialAcc	mgwrsar
SpatioTemporal	celltrackR	IFC	GmAMisc	sindyr	CAinterprTools
Survival	survivalMPLdc	BayesMixSurv	GofCens	straweib	PICBayes
TeachingStatistics	mdsr	fastR2	gpk	ADVICE	RSADBE
TimeSeries	WaveletArima	WaveletSVR	WaveletRF	TSF	WaveletANN
Tracking	flightsbr	FASeg	PhysActBedRest	pooling	irg
WebTechnologies	rdian	AzureCognitive	RPublica	tinyspotifyr	hakaiApi

Table 8: Top 5 packages with largest predicted probabilities within each Task View. In decreasing order from left to right. Using snapshot taken on the 4th of April 2022

8 Supplemental Material

Alongside this report I have uploaded a self-contained folder containing R code and data to create all of the figures created in this report. This folder has a R project file (`.Rproj`) that can be used to set the working directory, therefore the code should run wherever the folder is stored. In order to do this, whenever running script first open the `.Rproj` file. To make sure the working directory is set correctly in RStudio there is the option to set the working directory as the project directory. The folder also contains all of the code regarding the analysis of the text data and the construction of the multinomial regression model.