1 Enabling Text Analytics

- Greeting
- Self
- Project
- Supervisor

2 Introduction

- The aims of this project were … (from slide)

"… attractive output quickly"

- in the form of charts and tables

"… range of text formats"

- as may occur in novels and free-response survey data, among others

"… with a similar user base"

- so for the sake of compatibility, must be built with the statistical programming language `R`, and the well-regarded `Shiny` package in R for creating interactive web-apps.

segue To motivate what and how, let's ask, "why?"

3 Why Perform Text Analysis?

(from slide)
"… patterns …"

- such as common phrases and keywords

"structure"

- especially in files with no inherent structure

"emotional"

- following the emotional trajectory of a novel, or the tone of free-form responses in a survey

"summaries"

- for when we just want the spark-notes version "differences"

- see the "outliers" in text, so to speak.
- I'll demonstrate that the application can perform all of the above, with the novel *Alice in Wonderland* serving as the text to analyse.

segue Let's consider the app itself

4 App Overview

- The program takes the form of a web application, allowing it to run in a browser
- It consists of two tabs. The first is processing, where the user can dictate the tidying to be performed on the text data. The next is visualisation, where the desired analysis is declared and the output is generated
- Mouse-driven, simple and intuitive to use
- Follows a tidy framework with text represented in tabular form, specifically, . . . (read from slides)

segue We will now look at the program in more detail

5 Text Preparation

- The first part of every analysis is processing — text is especially messy data
- The application provides the ability to simplify words to their dictionary form, so we can compare like with like, in a process called *lemmatisation*
- We can also remove words that create more noise than signal. Words like "the", "a", or "and". These are called *stop-words*, and are removed in a dictionary method by matching them up to a stop-word lexicon, with a few different lexicons provided in-app, with a drop-down allowing user choice of lexicon.

segue Once processing is complete, we can move on to the visualisation tab

6 Visualisation

- The section asks the user (from slide) “What do you want to see, and how do you want to see it?”
- What you want to see is the output of an *insight measure*, which is a particular analysis performed on the text. In this case, the default is term frequency, the count of how many times each word appears in the text.
- “How to see” refers to the visualisation of that *insight measure*. Here, it is through a bar plot, but the user can change this by a drop-down list to any other visualisation. For example, a word-cloud, as demonstrated in the following video (▷ ▷ ▷)
- The size of each word in a word-cloud is proportional to it’s frequency
- We can see there is a lot of talking, and a character named “Alice”

segue But, how do these words relate to each other?

7 N-grams

- We can begin to answer that by looking at n-grams. N-grams are (from slides) sequences of n words occurring in order in a text. The insight measure can be changed by a drop-down list like so (▷ ▷ ▷ continue talking during playing) and n-grams are computed. N can be changed by dragging a slider.
- We can see that the most common 2-grams relate to the characters in the story, as well as the distributor, *Project Gutenberg*

segue Interestingly, the most frequent 4-grams showed repetitive phrases from songs in the book. To find important sentences though, there is a better way.

8 Key Sentences

(from slides)

- We ran LexRank on it’s own introduction paper, generating a surprisingly good summary of itself in just 5 sentences.

segue We aren’t limited only to sentences

9 Key Words

(from slides)

- Key words are similar, but different to the highest frequency words — often the most frequent words are localised in a text and not representative of the text as a whole

segue And considering that the whole is not always the sum of it's parts, we should also consider analyses in the aggregate

10 Distribution of Sentence Lengths

- Here is an aggregating function looking at the number of words within some aggregating variable. In this case, the variable is sentences
- The default visualisation is a histogram, showing a fairly interesting distribution

segue It would be worthwhile aggregating this further, seeing how the distribution changes throughout sections of the text

11 Sectioning

- For that to take place, further processing must be done
- Plain-text files are unstructured in the absolute, and the application includes a tool that will infer sections based on user selection, and create an aggregating variable — here, we'll select “chapter”

segue Switching back to the visualisation tab ...

12 Grouping

- We can now look at the sentence lengths of each chapter, this time visualised by density estimate plots for the sake of comparability
- Note here that the sectioning has picked up a chapter 0, which is the long legal preamble regarding licensing of the work — the story begins with chapter 1.
- That zeroth chapter is interesting in it's own right, bearing a different distribution to the rest of the novel, reflecting that it is just legal language regarding the licensing of the work, with separate authorship to the writer of *Alice in Wonderland*, Lewis Carroll. This same method

of looking at sentence length distributions has been used to cast doubt on the authorship of some of Shakespeare's plays, stirring up some very interesting controversy

- We can see here that the story starts slow, with long sentences in chapter 1. A quicker pace develops with the shortest sentences occurring in the middle, returning to normality toward the end, straight out of a *Hero's Journey*

segue It is also worth exploring how all of this is coloured by the changing mood of the story

13 Moving Average Term Sentiment

- Each word of the story can have some number assigned to it by a dictionary method to indicate the direction and magnitude of it's positive or negative emotional charge. There exist many different dictionaries, or "sentiment lexicons", with some provided in the app.
- The visualisation here is a time series plot of the moving average of word sentiment, with a variable window length controlled by a slider as an option.
- It is worth noting that dictionary methods are not the only means of determining sentiment, with more advanced models also capable of inferring sentiment and dealing with the complexities of words in context including situations involving negation, amplification, and the like.

segue It would be interesting to know what that trough is at the $\frac{2}{3}$ mark …

14 Split by Chapter

- And sectioning as before, we can see that the trough of negative sentiment occurs in chapter 9, which corresponds to a game of croquet with the Queen gone terribly wrong, involving demands for beheading — it seems that the sentiment analysis picked that up quite well
- This splitting and grouping is also especially useful for free-response survey data. An example data-set we used in testing the app was a portion the New Zealand Quality of Healthcare Survey, and we found some distinct patterns when grouping by individual doctors and assessing the distribution of sentiments relating to each of them.

segue So we have managed to briefly pick apart the novel with the application. Let's have a look at how I made the app.

15 Program Architecture

(from slides)

- The shiny app is decoupled into server and UI functions, controlling run-time instructions and appearance, respectively

“Near-purely functional”

- In the programming paradigm sense of the word
- The package maintains high procedural and functional cohesion within the processing, insight, and visualisation sections. The server in the app is intelligent enough to automatically match insight functions with visualisations, so the user only has to maintain the conception of processing and visualisation
- In all, the program makes use of about 1200 expressive lines of code
- In terms of external `R` libraries, the app is standing on the shoulders of giants, such as `ggplot` for the visualisations, but we have been trying to minimise dependencies, after a few bad encounters involving packages updating and not maintaining backwards compatibility

segue The program is extremely modular, allowing plenty of room for future work through it’s generality

16 Future Development

(explain sub-items)

17 Conclusion