Contents

1 Meeting With Chris **2**

2 Initial Survey **2**

2.1 Prescribed Reading 2
2.2 Consideration of Application Features 2
2.3 Twitter 3
2.4 Subtitles 3
2.5 R Packages 3
2.6 Other Text Analytics Applications 5
2.7 Scope Determination 5

3 Features **6**

3.1 Introduction 6
3.2 Processing 6
3.3 Within-Text Analytics 7
3.4 Between-Text Analytics 8
3.5 Stopwords 8
3.6 Visualisation 9
3.7 Text Summarisation 9
3.7.1 TextRank 11
3.7.2 LexRank 11
3.8 Search Function 11
3.9 Topic Modelling 11
3.10 Sentiment Distribution 12
3.11 Conditional Analytics 12
3.12 Wrapper Functions 12
3.12.1 Text Summarisation 12

4 Data Types **13**

4.1 Test Corpus 13
4.2 Free-Response Data 14
4.3 Data types for implementation 14

1 Meeting With Chris

2 Initial Survey

2.1 Prescribed Reading

- Text Mining with R: notes
- Text Analysis with R

2.2 Consideration of Application Features

UX/UI must be intuitive, modelling the inherently non-linear workflow, while fitting in with the iNZight / lite environment. I wrote some notes on UX/UI here.

Text Classification seems to be a hot topic in sentiment analysis, but the question remains of whether it is within our scope (I suspect not). If it were, openNLP, along with some pre-made models, would likely serve this topic well. An interesting example of text classification in the Dewey Decimal System is given here.

Zipf's Law is referred to regularly in text analysis. Should we demonstrate this directly, as part of some other analysis, or not at all?

The **form of analysis** will vary enormously with different forms of text. There are some things constant for all forms of text, but a good deal is very specific. For example, the information of interest will differ between novels, discourse (interviews, plays, scripts), twitter posts, survey response data, or others. It may be worthwhile to either focus solely on one, or give the option to specify the type of text.

Data structures will change based on the text type, and the packages used. With a tidy dataframe as our base structure, it is easy enough to convert to specific objects required by various packages, harder to convert back.

There is a natural link between **text analysis and Linguistics**, and a significant amount of the terminology in the field reflects that. Our application requires far more market share than one that a mention in a third year linguistics paper provides, and so linguistics is not going to be our primary focus. Regardless, many forms of analysis require some linguistic theory, such as those dependent on Part of Speech tagging, so it is still worthwhile to keep in mind

2.3 Twitter

Twitter data looks particularly interesting, as it is a constantly updating, rich source of information. I wrote up some notes on text mining twitter here. It would be particularly interesting to view twitter data in the context of discourse analysis.

2.4 Subtitles

Subtitles are a unique form of text that would be very interesting to analyse. Subtitles for films and TV Series can be obtained easily from the site opensubtitles, though obtaining subtitles programatically may be more difficult. It clearly is possible, as VLC has an inbuilt feature, as does subsync, which is written in C#, so would require a port to R (probably not worth it for us at this point). Subtitles usually come as .srt files, and once the file is obtained, it's easy enough to import and work with it in R with the package subtools.

2.5 R Packages

Here is a useful comparison between the major text mining packages. CRAN also has a task view specifically for Natural Language Processing, offering many packages relevant to this project. Interestingly, they are split by linguistic category; Syntax, Semantics, and Pragmatics. The further from syntax the package is, the far more interesting it intuitively appears (eg. word count vs sentiment analysis). Some packages of interest include:

tidytext is a text-mining package using tidy principles, providing excellent interactivity with the tidyverse, as documented in the book Text Mining with R

tm is a text-mining framework that was the go-to for text mining in R, but appears to have been made redundant by tidytext and quanteda of late

quanteda sits alone next to qdap in the Pragmatics section of the NLP task view, and offers a similar capability to tidytext, though from a more object-oriented paradigm, revolving around *corpus* objects. It also has extensions such as offering readability scores, something that may be worth implementing.

qdap is a "quantitative discourse analysis package", an extremely rich set of tools for the analysis of discourse in text, such as may arise from plays, scripts, interviews etc. Includes output on length of discourse for agents, turn-taking, and sentiment within passages of speech. This looks to me like the most insight that could be gained from a text.

sentimentr is a rich sentiment analysis and tokenising package, with features including dealing with negation, amplification, etc. in multi-sentence level analysis. An interesting feature is the ability to output text with sentences highlighted according to their inferred sentiment

dygraphs is a time-series visualisation package capable of outputting very clear interactive time-series graphics, useful for any time-series in the text analysis module

gganimate produces animations on top of the ggplot package, offering powerful insights. Here is an example demonstrating Zipf's Law

textrank has the unique idea of extracting keywords automatically from a text using the pagerank algorithm (pagerank studied in depth in STATS 320) - my exploration of the package is documented here

Packages for obtaining text:

gutenbergr from Project Gutenberg

rtweet from Twitter

wikipediar from Wikipedia

ggpage produces impressive page-view charts with features such as word highlighting, allowing for a clear overview of a text and it's structure, with probable use in our search feature function

gganimate produces animated charts, which can be useful if additional, regular, and low *n* dimensions exist in the data

---

Additionally, there are some packages that may not necessarily be useful for the end user, but may help for our development needs. These include:

- udpipe performs

tokenisation, parts of speech tagging (which serves as the foundation for textrank), and more, based on the well-recognised C++ udpipe library, using the Universal Treebank

- BTM performs Biterm Topic Modelling,

which is useful for "finding topics in short texts (as occurs in short survey answers or twitter data)". It uses a somewhat complex sampling procedure, and like LDA topic modelling, requires a corpus for comparison. Based on C++ BTM

- crfsuite provides a modelling

framework, which is currently outside our current scope, but could be useful later

- In the analysis / removal of names, an important component of a text,

humaniformat is likely to be useful

- CRAN Task View: Web Technologies and Services for importing texts from the

internet

2.6 Other Text Analytics Applications

The field of text analytics applications is rather diverse, with most being general analytics applications with text analytics as a feature of the application. Some of the applications (general and specific) are given:

- txtminer is a web app for analysing text at a deep level (with something of a linguistic focus) over multiple languages, for an "educated citizen researcher"

2.7 Scope Determination

The scope of the project is naturally limited by the amount of time available to do it. As such, exploration of topics such as discourse analysis, while interesting, is beyond the scope of the project. Analysis of text must be limited to regular texts, and comparisons between them. The application must give the greatest amount of insight to a regular user, in the shortest amount of time, into what the text is actually about.

Cassidy's project was intended to create this, and I have written notes on it here.

Ultimately, I am not completely sold on the idea that term frequencies and other base-level statistics really give that clear a picture of what a text is about. It can give some direction, and it can allow for broad classification of works (eg. a novel will usually have character names at the highest frequency ranks, scientific works usually have domain specific terms), but I think word frequencies are less useful to the analyst than to the algorithms they feed into, such as tf-idf, that may be more useful. As such, I don't think valuable screen space should be taken up by low-level statistics such as term frequencies. To me, the situation is somewhat akin to Anscombe's Quartet, where the base statistics leave a good deal of information out, term frequencies being analogous to the modal values.

Additionally, sentiment is really just one part of determining the semantics of a text. I think too much focus is put on sentiment, which in practice is something of a "happiness meter". I would like to include other measurement schemes, such as readability, formality, etc.

Some kind of context in relation to the universal set of texts would be ideal as well, I think a lot of this analysis occurs in a vacuum, and insights are hard to come by - something like Google n-grams would be ideal.

I'm picturing a single page, where the analyst can take one look and have a fair idea of what a text is about. In reality it will have to be more complex than that, but that is my lead at the moment. With this in mind, I want to see keywords, more on *structure* of a text, context, and clear, punchy graphics showing not *just* sentiment, but several other key measurements.

3 Features

3.1 Introduction

The application essentially consists of a feature-space, with the area being divided in three; Processing, Within-Text Analytics, and Between-Text Analyticso. This follows the general format of much of what is capable in text analysis, and what is of interest to us and our end users. The UI will likely reflect this, dividing into seperate windows/panes/tabs to accomodate. Let's look at them in turn:

3.2 Processing

In order for text to be analysed, it must be imported and processed. A lot of this is an iterative process, coming back for further processing after analysis etc. Importing will have a "type" selection ability for the user, where they can choose from a small curated list of easy-access types, such as gutenberg search, twitter, etc. The option for a custom text-type is essential, allowing .txt, and for the particularly advanced end-user, .csv.

Once the file is imported/type is downloaded, the option should exist to allow the specification of divisions in the text. In a literary work, these include "chapter", "part", "canto", etc. A twitter type would allow division by author, by tweet, etc. An important aspect of this processing is to have a clear picture of what the data should look like. Division of a text should be associated with some visualisation of the resulting structure of the text, such as a horizontal bar graph showing the raw count of text (word count) for each division - this would allow immediate insight into the correctness of the division, by sighting obvious errors immediately, and allowing fine tuning so that, for example, the known number of chapters match up with the number of divisions. We could implement a few basic division operators in regex, while following the philosophy of allowing custom input if wanted. Example regex for "Chapter" could be `/[Cc]hapter[.:]?[ ]{0,10}-?[ ]{0,10}([0-9]|[ivxIVX]*))/g`, something the end user is likely not wanting to input themselves.

Removal and transformation is another important processing step for text, with stopwords and lemmatisation being invaluable. The option should exist to remove specific types of words, which can again come from prespecified lists. An aspect worth considering is if this should be done in a table manipulation, or a model - or both, with the length of the text deciding automatically based on sensible defaults. Again, the need for a clear picture of the data is essential, with some visual indication of the data during transformation and removal essential; this could take the form of some basic statistics, such as a ranking of terms by frequencies, and some random passage chosen.

Processing multiple documents is also essential. The importation is something that has to be got right, otherwise it'll be more complex than it already is, and the end-user will lose interest before the show even begins. My initial thoughts are of a tabbed import process, with each tab holding the processing tasks for each individual document, however this won't scale well to large corpus imports.

3.3 Within-Text Analytics

Within-text analytics should have options to look at the whole text as it is, whether to look by division, or whether to look at the entire imported corpus as a whole.

A killer feature here is the production of a summary; a few key sentences that summarise the text. It's a case of using text to describe text, but done effectively, it has the potential to compress a large amount of information into a small, human-understandable object.

Related to the summary, keywords in the text will give a good indication of topics and tone of the text, as well as perhaps more grammatical notions, such as authorial word choices. There is the possibility of using keywords as a basis for other features, such as the ability to use a search engine to find related texts from the keywords.

Bigrams and associated terms are also excellent indicators of a text. Something I particularly liked in Cassidy's project was the ability to search for a term, and see what was related to it. In that case, the text was "Peter Pan", and searching for a character's name yielded a wealth of information of the emotions and events attached to the character.

Sentiment is a feature that has been heavily developed by the field of text analytics, seeing a broad variety of uses. here, it would be worth examining sentiment, by word and over the length of the text overall.

3.4 Between-Text Analytics

As in within-text analytics, between-text analytics should have options for specifying the component of the text that is of interest; here, the two major categories would be comparisons between divisions within an individual text, and comparisons between full texts.

Topic modelling gives an idea of what some topics are between texts - something odd to me is that there isn't a huge amount of information on topic modelling purely within a text, it always seems to be between texts (LDA etc.)

tf-idf for a general overview of terms more or less unique to different texts.

Summarisation between all texts would also be enormously useful.

3.5 Stopwords

After noting that stopword removal impacted important n-grams when a stopword made up some component of the n-gram, it becomes very worthwhile to not only include an active capacity to view what current stopwords exist, but also to have alternative lists of stopwords. The following summarises some research into stopwords and common practices around them;

- StackOverflow removes the top 10,000 most common english words in "related queries" for their SQL search engine (`https://stackoverflow.`

`blog/2008/12/04/podcast-32/`)

- The stopwords `R` package includes several lists of stopwords. Among these, of note are:
  - SMART: The stopword lists based on the SMART (System for the Mechanical Analysis and Retrieval of Text)Information Retrieval System, an information retrieval system developed at Cornell University inthe 1960s.
  - snowball: It is a small string processing language designed for creating stemming algorithms for use in Information Retrieval.
  - iso: The most comprehensive stopwords for any language

The package we are using extensively, tidytext, has both SMART and snowball lists, as well as onix, which bills itself as " probably the most widely used stopword list. It covers a wide number of stopwords without getting too aggressive and including too many words which a user might search upon." Of note is that all of the lists are included in one dataframe, so it should be filtered before being used, unlike how we have been using it. snowball is clearly the shortest, and I think may be worth having as the default, with SMART (the most extensive) and onix as secondary options. We are not in the role of providing a computationally efficient search engine, only removing words that contribute little but noise.

In terms of implementation within our program, we ought to have the ability to add custom stopwords. In keeping with the philosophy of having our data clearly visible, this will necessitate a "temporary stopwords" list. In the process of implementation, we will have to make assesments of whether it will run too slowly if allowed to influence charts and output in real timme, so manual refreshes would be required. Additionally, it will be good to have a running set of statistics keeping available what has been done to the data (including more than just stopword removal)

3.6 Visualisation

With so much of the conceptual space of text analytic visualisation being taken up with far from optimal charts, there is a need to experiment with alternative visualisations; We explore some here

3.7 Text Summarisation

Wikipedia: Automatic Summarisation

Text summarisation creates enormous insight, especially from a long text. There are a variety of different techniques, of varying effectiveness and efficiency. A famous example of automatic text summarisation comes from autotoldr, a bot on reddit that automatically generates summaries of news articles in 4-5 sentences. Autotldr is powered by SMMRY, which explains it's algorithm as working through the following steps:

1. Associate words with their grammatical counterparts. (e.g "city" and "cities")
2. Calculate the occurrence of each word in the text.
3. Assign each word with points depending on their popularity.
4. Detect which periods represent the end of a sentence. (e.g "Mr." does not).
5. Split up the text into individual sentences.
6. Rank sentences by the sum of their words' points.
7. Return X of the most highly ranked sentences in chronological order.

The two main approaches to automatic summarisation are extractive and abstractive; **Extractive** uses some subset of the original text to form a summary, while **abstractive** techniques form semantic representations of the text. Here, we will stick to the clearer, simpler, extractive techniques for now.

textrank has the unique idea of extracting keywords automatically from a text using the pagerank algorithm (pagerank studied in depth in STATS 320) - my exploration of the package is documented here. At present, the R implementation of it creates errors for large text files, but it is worth exploring more into it - whether it is the implementation, or if it is the algorithm itself.

Hvidfeldt is a prolific blogger focussing on text analysis - he put up this tutorial on incorporating textrank with tidy methods: tidy textRank

Further summarisation experimentation is continued here

After further testing, I have found LexRank to work significantly faster, while generating similar results, thus being favourable for summarisation. It appears that Textrank wins in the ability to generate keywords, and does so extremely quickly. Despite the speed gain in using LexRank for summarisation, it still takes several seconds on my i5 dual-core, to run, however this is offset by the verbosity of the function assuring me that it isn't hanging.

LexRank and textRank appear to exist complimentarily to one another. Below is a brief summary of how they work

3.7.1 TextRank

TextRank essentially finds the most representative sentence of a text based on some similarity measure to other sentence.

By dividing a text into sentences, measures of similarity between every sentence is calculated (by any number of possible similarity measures), producing an adjacency matrix of a graph with nodes being sentences, edge weights being similarity. The PageRank algorithm is then run on this graph, deriving the best connected sentences, and thereby the most representative sentences. A list is produced giving sentences with their corresponding PageRank. The top n sentences can be chosen, then output in chronological order, to produce a summary.

In the generation of keywords, the same process described is typically run on unigrams, with the similarity measure being co-occurance.

3.7.2 LexRank

LexRank is essentially the same as textRank, however uses cosine similarity of tf-idf vectors as it's measure of similarity. LexRank is better at working across multiple texts, due to the inclusion of a heuristic known as "Cross-Sentence Information Subsumption (CSIS)"

3.8 Search Function

The analyst is not expected to be entirely familiar with the texts under analysis; this is partly the purpose of this program. Hence, there are likely to be terms, keywords, and relationships that the program reveals, and are a surprise to the analyst, and context is necessary to understand them. A search function has been identified as useful in meeting this problem, where a word is entered in search, and contextual passages are returned. Useful in the results would be indications of location of each passage in the greater text, as well as if multiple texts are present, the name of the text it belongs to.

3.9 Topic Modelling

Topic Modelling appears to serve a useful purpose in text analytics, with LDA being the primary implementation, requiring multiple texts, and a

Document-Term Matrix. My exploration with topic modelling is located here. It could be worth investigating other forms of topic modelling, especially within-text.

[2019-05-17 Fri] I checked other forms - their complexity requires a great deal of time to understand if I want to implement them intelligently; better to stick with LDA, which, while also complex, is well used enough to be considered standard.

3.10 Sentiment Distribution

Over a large n dataset such as free-response surveys, it may be useful to calculate the sentiment for each response, and consider the statistical properties of the distribution of sentiments. Here is an exploration of free-response data forming a sentiment distribution.

3.11 Conditional Analytics

The idea of conditional analytics is of interest to me, especially for high n datasets such as large free-response surveys. Particularly, I want to know, given some condition, how does the subset behave? For example, given a negative sentiment, what is the most representative response? Or, given that some common word, what is the distribution of sentiment

3.12 Wrapper Functions

In order to begin implementation, I have defined wrapper functions for the primary features. The intention is to create a higher layer of abstraction for the features as well as ease of use. I begin with the text summarisation feature; the details are below

3.12.1 Text Summarisation

Link to src Link to test Arguments:

- x = input dataframe with column titled "word"
- n = n-many sentences
- style = style of output (chart, dataframe, text)
- dim = dimension of chart
- engine = textrank/lexrank

- type = sentences/keywords etc.

Working through, I have come to come realisation that a complete wrapper function may not necessarily be ideal; rather, a pipeline may be better - this is because a wrapper function, with, e.g., a plotting function at the outermost layer, would require a full recalculation of the inner functions for every parameter change in the plot - what may be better is the creation of a pipeline that leaves most functions as they are but just creates more suitable objects to pass as arguments to the functions. This is something of a "memory cheap; processing expensive" principle. The display wrapper functions would then be taking complete objects only

<2019-05-22 Wed> Chris clarified the role of wrappers here being more of a "layer" level, layers being:

- word/n-gram;
  - Word frequency
  - Bigram frequency
  - pairwise word correlations
  - textrank keywords
- sentence;
  - textrank
  - lexrank
- topic level
- sentiment level

4 Data Types

The application requires the capacity to smoothly work with diverse data types. For this to occur, a test corpus must be developed, and some important data types picked out.

4.1 Test Corpus

It is essential to test on a broad variety of texts in order to create the most general base application, so a "test set" will have to be developed. All data is stored in the folder data

Must have

- Literature (eg. Dante's Divine Comedy)
- Survey response data (eg. nzqhs, Cancer Society)
- Transcript; lack of punctuation may cause difficulties in processing sentences.
- Twitter

Would be nice

- article
  - journal (scientific, social)
  - news
  - blog
  - wikipedia
- discourse
  - interview
  - subtitles
- documentation
  - product manual
  - technical user guide

4.2 Free-Response Data

Free Response Data (as in survey forms etc.) has been identified as an area of high potential for the application. Two datasets have been used to run typical text analyses upon, with the exploration here. Upon close inspection, there are subtleties worth exploring further especially in bigrams and keywords.

4.3 Data types for implementation

In the production of wrapper functions, we require data types that work well with all functions that are required. For the purpose of word-level summarisation, the following features require functions with the associated data types as arguments:

- Word frequency: `tidytext::unnest_tokens`

    - @param tbl: A data frame

- Bigram frequency: `tidytext::unnest_tokens`
    - @param tbl: A data frame
- pairwise word correlations: `widyr::pairwise_cor`
    - @param tbl: Table
    - @param: item: Item to compare; will end up in 'item1' and 'item2' columns
    - @param feature: Column describing the feature that links one item to others
- textrank keywords: `textrank::textrank_keywords`
    - @param x: a character vector of words.