Text Mining Analysis of 'The Hobbit' by J.R.R. Tolkien

1. Introduction:

Text Mining is a technique of exploring unstructured text data usually in large volumes to identify the patterns for turning it into meaningful and actionable information. We use machine learning algorithms and apply techniques from statistics, data mining, and computational linguistics. The 3 basic steps of Text mining are data collection, data pre-processing, and data analysis. The pre-processing step involves the removal of irrelevant information for the analysis. The goal of the pre-processing step is to convert the unstructured raw data into a structured format without losing accuracy. The text mining step identifies and extracts the patterns and trends in the text document.

2. Methods & analysis

2.1. Libraries used

The tm package (Text Mining Package) which contains all necessary functions for text data import, pre-processing, corpus handling, metadata analysis, vector source interpretation and creation of term-document matrices. The main structure for managing documents in tm package is *Corpus*, representing a group of text documents that has specific attributes for performing certain types of analysis. It exists in two ways: Volitile Corpus (VCorpus) is a temporary object that is stored in memory and gets deleted when the R object containing it is destroyed. Permanent Corpus (PCorpus) is a permanent object that stores documents outside of R in a database.

The SnowballC package is an alternative interface to a C version of Porter's word stemming algorithm for collapsing words to a common root to aid comparison of vocabulary. Text stemming is a technique used to extract the base form of the words by removing affixes from them thereby aiding in the pre-processing of text, words, and documents for text normalization.

The wordcloud package has functions to create colourful word clouds, to compare and contrast the texts in the document, and to avoid over-plotting while creating a text based scatter plots. The RColorBrewer package offers several color palettes for creating colourful word clouds.

2.2. Loading the dataset

The dataset provided for this assignment is *the_hobbit.txt* is a standard text document that contains plain text. The file contains the text of a 1937 novel, “The Hobbit” written by J. R. R. Tolkien. It has 19 chapter and 6424 lines. We use the function setwd() to change the current directory to the directory in which the text file is stored and it is assigned a variable *filePath*. We then use the readLines() function to read all the text in the variable *filePath* which is stored in the new variable *text.* We pass the logical parameter FALSE to the attribute *warn* for avoiding missing EOL(end of line) warnings on the last line.

3. Data Pre-processing

3.1. Loading a corpus

Corpus is the structure that stores the collection of text in which we perform our text mining analysis. The package tm supports variety of sources which can be listed by the getSources() function. We use vector source which interprets each element of the vector *text* as a document. We then load the document as corpora by using the Corpus() which has the collection of the text in the *the_hobbit.txt* and it is stored in a variable called *docs*.

We use the inspect() function to check whether the text in the document has been loaded properly. The output of inspect(text) list out the text with the line number as it’s indices, which in our text file is 6424 indices.

3.2. Getting rid of the symbols

We use the function *gsub()* which looks for exact match of all the character given to argument *pattern* within each element in the corpus document. The argument *x* represents the character vector in which the match is sought. The space " "is passed as a parameter for the argument *replacement*. We also use the content_transformer() to create our own function to modify the text content, which in this case is to search for the pattern and replace it with the blank space. Finally, we use the tm_map() which is the transformation function for applying it on the structure corpora. The corpus *docs* is passed as an parameter in the argument x and the custom transformation function *toSpace* is passed as an parameter for the argument *FUN*. The symbols "/”, "@”, "\\|" are used as an argument for the custom function *toSpace* which replaces it with a space.

Even though the symbols "/”, "@”, "\\|" are not found in the text file *the_hobbit.txt,* it’s a general practice to look for specific symbols in the text file and remove them. For example, while text mining the data with email addresses, removing the symbol @ is recommended.

3.3. Transforming to lower case

In the text file, some words are found to be either in lower case or as a capitalized word. So, we use the tolower() which is a character translation function for converting upper case to lower case. Since it’s not a "canonical" transformation, we need to use it along with the content_transformer() function. Finally, we use the tm_map() which applies these transformation in the corpora *docs*.

3.4. Removing the numbers

In the text file, we found the use of numbers in many places which may not be relevant for our analyses. So, we use the function *removeNumbers* to remove the numbers from the text file. We wrap the above function with tm_map() to applies these transformation in the corpora *docs*.

3.5. Removing English Stop Words

The stop words are the commonly used words in English which can be eliminated while text mining as they carry very little useful information. Some of the examples of the common stop words are a, for, the, very, is, further, of, between, more, etc. we use the function *removeWords* to remove words from a text document and we pass *stopwords("english")* containing 174 stop words which is provided by tm package. We wrap the above function with tm_map() to applies these transformation in the corpora *docs*.

3.6. Getting rid of the symbols

In addition to the above stop words, we found few words in the text file which provides very less useful information. We found the words "said" and "like" are used multiple times and doesn’t hold much information. To remove these words, we use the same *removeWords* function and pass the character vector c("said", "like") as it's argument. We wrap the above function with tm_map() to applies these transformation in the corpora *docs*.

3.7. Removing Punctuation

In the text file, we found the use of many punctuations which provides grammatical context for understanding the text. Since we are interested in creating the word cloud, we prefer to remove the punctuations from the corpus. We use the removePunctuation() to remove punctuation marks from a text document.

3.8. Removing Whitespace

In the text file, we found multiple whitespace characters which needs to be collapsed to a single blank. We use the *stripWhitespace* function to remove these whitespace which is wrapped with tm_map() to applies these transformation in the corpora *docs*.

3.9. Test Stemming

Text stemming is a text normalization technique used to extract the base form of the words by removing affixes from them. The stemDocument() function is applied for this purpose uses the Porter's stemming algorithm.

4. Creating a Term Document Matrix

A term-document matrix is a mathematical matrix that describes the frequency of the words in the corpus file. It uses terms as the rows and documents as the columns and a frequency of the words as the cells of the matrix. We use TermDocumentMatrix() function to construct the matrix.

```
> dtm
<<TermDocumentMatrix (terms: 11641, documents: 6424)>>
Non-/sparse entries: 72627/74709157
Sparsity           : 100%
```

Figure 1. The output displays the term-document matrix

In the fig.1, the *terms* represent the number of unique words and the *documents* represent the number of sentences in the text file. Sparsity is the percentage of sparse entries in the entire matrix. Since the sparsity is 100% in our output, we can say that the matrix is fully spared which means mostly empty. The number of terms in the longest document in the corpus docs is referred as maximal term length.

4.1. Converting the Term Document Matrix to a simple matrix

The term-document matrix is not reader-friendly for most software. So, we need to convert the sparse matrix into a normal R matrix. We use the as.matrix() function to convert it into a standard matrix which can be read as csv file. The dimension of the matrix *m* is 11641 rows and 6424 columns.

4.2. Sorting the words in the matrix by frequency

We use the rowSums() to calculate the sum of the terms in the row of a matrix. We pass the logical parameter TRUE for the argument *decreasing* to change the order from ascending to descending and it is applied using the sort() function.

```
> v
          the         and         was        they        that
         6040        4392        1348        1334         955
          had         his         for         not        were
          942         902         694         692         684
          you        with         but         all       their
          663         642         587         578         525
         said       there        have        from       bilbo
          476         458         433         405         384
        could        them         out         are        down
          348         336         312         296         270
         when         one         him        came       would
          264         253         248         246         244
         what        into        very        then         now
          243         242         239         232         228
         this        like     dwarves        more        been
          224         219         212         210         204
       before        long        some        your        will
          199         197         196         196         193
```

Figure 2. The output displays the variable ‘v’ sorted by word frequency

In the fig.2, we can observe that 11641 unique words are listed with their word frequency in descending order. The word ‘the’ has the most frequency of 6040 followed by the words ‘and’ and ‘was’.

4.3. Creating the dataframe with word frequency

We use the data.frame() to create a dataframe of word frequency with 2 columns – word and frequency. The parameter *names(v)* which contain the 11641 unique words in the text file is passed in the argument *word.* The variable ‘v’ which is in numeric class, contains the count of each unique words is passed as a parameter for the argument *freq*.

```
> names(v)
 [1] "the"     "and"     "was"     "they"
 [5] "that"    "had"     "his"     "for"
 [9] "not"     "were"    "you"     "with"
[13] "but"     "all"     "their"   "said"
[17] "there"   "have"    "from"    "bilbo"
[21] "could"   "them"    "out"     "are"
[25] "down"    "when"    "one"     "him"
[29] "came"    "would"   "what"    "into"
[33] "very"    "then"    "now"     "this"
[37] "like"    "dwarves" "more"    "been"
[41] "before"  "long"    "some"    "your"
[45] "will"    "great"   "about"   "did"
[49] "come"    "after"   "still"   "back"
[53] "only"    "which"   "little"  "far"
[57] "went"    "time"    "even"    "over"
[61] "any"     "than"    "last"    "see"
```

Figure 3. The output displays the names() of the variable ‘v’

From fig.3, we can observe that the names(v) contain the 11641 unique words in the text file *the_hobbit.txt*

```
> d
      word freq
the    the 6040
and    and 4392
was    was 1348
they  they 1334
that  that  955
had    had  942
his    his  902
for    for  694
not    not  692
were  were  684
```

Figure 4. The output displays the dataframe ‘d’

From fig.4, we can observe that the datframe ‘d‘ contain the 11641 rows and 2 columns. The column word represents the unique words and the column freq represents their word count.

4.4. Generating Word Cloud

A word cloud is a visual representation of text data which depict keyword based on it's frequency. In R, the wordcloud package has the functions for creating the word cloud. We use the function *wordcloud()* for generating the word cloud. The word column(d$word) is used as a parameter for the argument *words.* The freq column(d$freq) is used as a parameter for the argument *freq.* We use 1 for the argument *min.freq* for plotting every word in the word column. We use 200 for the argument *max.words* which is the maximum number of words to be plotted. We pass the logical parameter FALSE for the argument *random.order* for plotting the words in decreasing frequency. We pass 0.35 for the argument *rot.per* which represents the proportion words with 90 degree rotation. We use the brewer.pal(8, "Dark2") as a parameter for the argument *colors* which represents the below colour palette

We use of set.seed() to obtain the same layout when generating the word cloud each time. Otherwise, the layout is randomly.

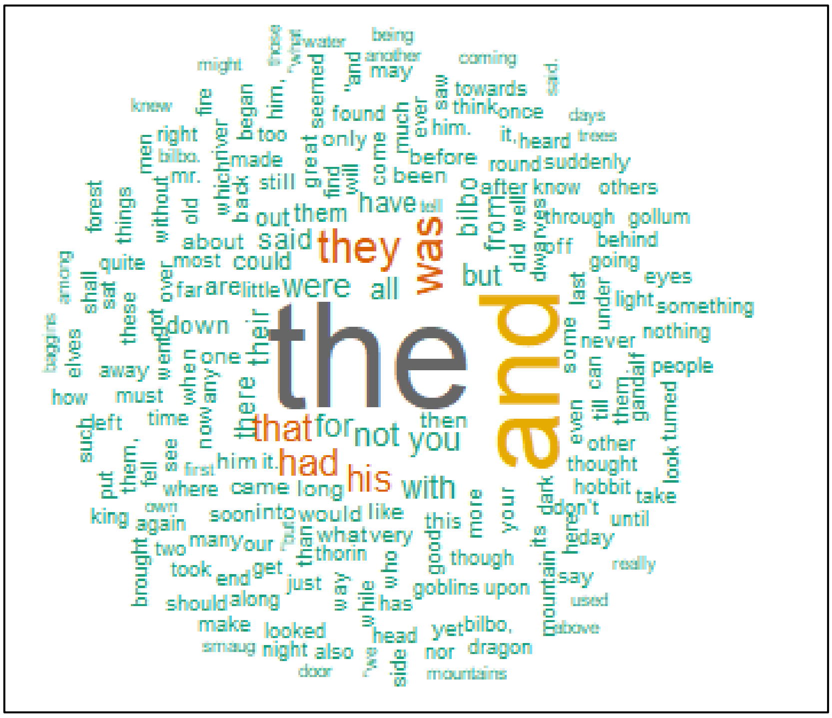

Figure 5. The output displays the word cloud

4.5. Identifying Frequent Items

To identify the most frequent terms in the corpus, we use findFreqTerms() which finds frequent terms in a term-document matrix.

```
> findFreqTerms(dtm, lowfreq = 100) #change to 4 if using MLK speech
 [1] "not"     "the"     "there"   "with"    "and"     "down"    "that"
 [8] "was"     "had"     "like"    "round"   "very"    "for"     "all"
[15] "but"     "into"    "little"  "many"    "out"     "one"     "then"
[22] "were"    "have"    "only"    "his"     "over"    "this"    "them"
[29] "time"    "any"     "never"   "they"    "could"   "did"     "what"
[36] "would"   "you"     "see"     "will"    "our"     "some"    "about"
[43] "are"     "than"    "when"    "which"   "can"     "come"    "their"
[50] "long"    "after"   "get"     "now"     "bilbo"   "old"     "who"
[57] "said"    "must"    "still"   "while"   "though"  "under"   "from"
[64] "got"     "just"    "more"    "came"    "been"    "way"     "him"
[71] "good"    "before"  "even"    "don't"   "your"    "goblins" "great"
[78] "went"    "off"     "far"     "soon"    "back"    "dwarves" "much"
[85] "made"    "last"    "thorin"  "through" "dark"    "where"
```

Figure 6. The output displays the 100 most frequent words

4.6. Finding the word association

We can also find the word association in the term-document matrixby using the function findAssocs(). The matrix is passed as an argument with correlation limit set as 0.1. The correlation limit is a measure of how closely associated the words are in the corpus. The limit of 1.0 display words which appear together and 0.0 display words that never appear together.

```
> findAssocs(dtm, terms = "gold", corlimit = 0.1)
$gold
              jewels             jewels,          steal
                0.18                0.18           0.17
          determined           lookishly   "undoubtedly
                0.15                0.15           0.15
              scarce          neck\xe2�"we  off"\xe2�here
                0.15                0.15           0.15
             stroked               coins         litter
                0.15                0.15           0.15
             gleamed               (made     chieftains
                0.15                0.15           0.15
              crown,     smith\xe2�craft,        things:
                0.15                0.15           0.15
            bar\xe2�                 due,            raw
                0.15                0.15           0.15
             silver,              fleets        waters,
                0.15                0.15           0.15
 mountain\xe2�gates,             rivers         costly
                0.15                0.15           0.15
```

Figure 7. The output displays the word association for the term ‘gold’

4.7. Plotting Word Frequencies

We can plot the most frequently used words as a barplot using the barplot(). We use the first 10 records of the datframe ‘d[1:10,]’ for the plot. The las argument is set as 2 to change the orientation perpendicular to the axis. We have used d[1:10,]$word for labelling the bars with the corresponding words.

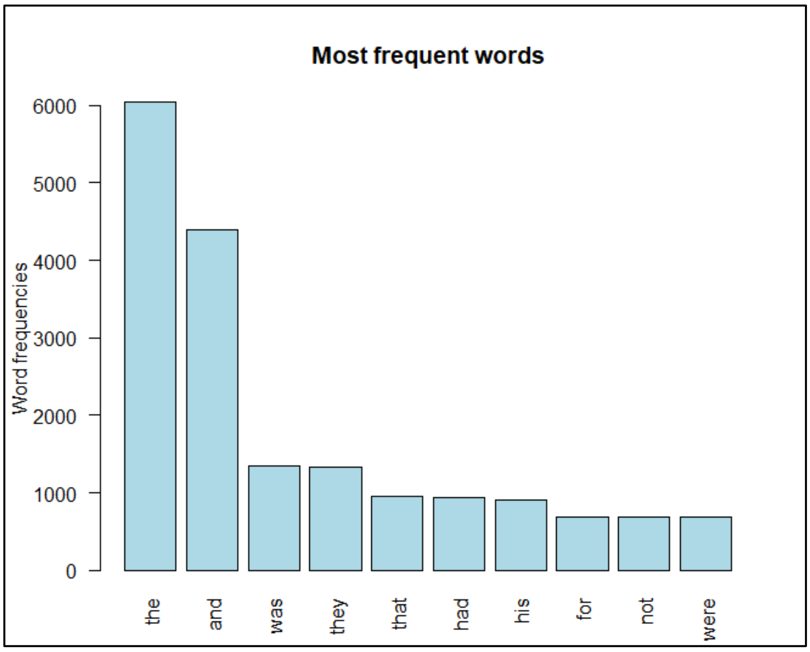

Figure 8. The output displays the bar chart of most frequently used words

From fig.8, we can say that the word ‘the’ has the highest frequency of 6000.

5. Recommendation

Based on the text mining analysis of "The Hobbit" text document, several recommendations emerge:

Refine Pre-processing Techniques: While the initial pre-processing steps effectively removed symbols, transformed text to lowercase, and eliminated numbers, further exploration could refine stop word removal and additional symbol handling to enhance data cleanliness.

Advanced Text Normalization: Experiment with more sophisticated techniques for text normalization, such as lemmatization, to ensure consistency and accuracy in word representation across the corpus.

Explore Advanced Analytics: Incorporate advanced analytics methods like topic modeling or sentiment analysis to derive deeper insights into the text, uncovering themes, emotions, or character sentiments.

Diversify Visualization Methods: While word clouds and bar charts offer intuitive insights, consider integrating other visualization techniques like network analysis to visualize word associations or word embeddings to capture semantic relationships.

Iterative Analysis: Text mining is an iterative process; continue refining techniques, experimenting with different parameters, and exploring alternative methodologies to extract more meaningful information from the text.

6. Conclusion

In conclusion, text mining provides a powerful framework for extracting valuable insights from unstructured text data. Through the analysis of "The Hobbit" text document, we demonstrated the application of various pre-processing techniques, creation of a term-document matrix, and generation of visualizations like word clouds and bar charts to understand word frequency and associations.

While this analysis provides a foundational understanding of the text, further refinement and exploration are encouraged to delve deeper into the nuances of the narrative. By leveraging advanced text mining methodologies and continuously iterating on the analysis, researchers and practitioners can unlock richer insights, enabling deeper comprehension and interpretation of textual data for diverse applications across domains.

7. References

1. Data Science Desktop Survival Guide. (n.d.). In onepager.togaware.com. Retrieved June 29, 2022, from https://onepager.togaware.com/index.html