Text Mining South Park

Dominik Nerger (i6146759)

June 5, 2017

Contents

1 Introduction

> And tell everyone in the past for us, that no one single answer... is ever the answer.
>
> UAA Leader, Ep. Go God Go XII

South Park is an animated series revolving around the four boys Stan Marsh, Kyle Broflovski, Eric Cartman and Kenny McCormick who live in South Park, Colorado. South Park has the third most episodes in regard to American animated TV series, totaling 277 episodes over 20 seasons.

Because of the huge amount of episodes, it is an interesting target for the practical assignment, which is part of the course *Information Retrieval & Text Mining*, with the possibility of acquiring new insights into a TV series that is used to satirize American culture.

The repository is available on GitHub[1].

2 Libraries

All scripts have been programmed in *R*. To execute the scripts, R needs to be installed. To view temporary files that are executed during runtime, e.g. the corpus or a TermDocumentMatrix, it is advised to install RStudio. The libraries necessary for each script are imported at the top of each script, if they are not installed they can be installed by executing:

```
install.packages("library-name")
```

In the following, all libraries that are related to Text Mining techniques will be introduced shortly.

The library **tm** is the Text Mining package of R, which enables pre-processing of data sets and allows to build the corpus. **RWeka** is a collection of machine learning algorithms for data mining tasks. **NMF** introduces the Non-negative Matrix Factorization to R. **NLP** and **OpenNLP** are libraries that provide Natural Language Processing techniques and are used for NER-Tagging. **syuzhet** extracts sentiments from text and contains the three sentiment dictionaries *bing*, *afinn* and *nrc*. The package **stm** is used for Structural Topic Modeling which is LDA with additional met-data and can be visualized using the package **LDAvis**. Libraries used for visualization include **igraph**,**ggplot2**, **ggraph**, **viridis** and **wordcloud**.

3 Data set

The data set spans from seasons **1** to **18**, adding up to 257 episodes overall with a file size of 5.41MB. It contains 70896 rows, with each row possessing

[1] `https://github.com/dnerger/South-Park-Text-Mining`

information about the **season**, **episode**, **character** and **line** and is available as a *.csv* file.

The data set has been crawled by Bob Adams and is available to download on GitHub[2]. It has been assembled by crawling the South Park Archives[3]. The code of this GitHub repository is not available to the public.

I made an attempt at a crawl with the R package **rvest**, which is able to harvest data from websites. However, the *html data* of the episode scripts provided on the South Park Archives differ between episodes. For 16 of 20 seasons, the whole episode needs to be pre-processed manually because the **line** and *character* are extracted individually and can not be merged because of random whitespace lines that are introduced by **rvest**.

In the scripts, the data set is used in three different variations:

1. *dialogue* - each entry holds information about season, episode, character and line
2. *by.season* - each entry holds information about season and the complete script for that season
3. *by.episode* - each entry holds information about that episode, season and script for that episode

Both *by.episode* and *by.season* are created by using the data from *dialogue*.

[2] `https://github.com/BobAdamsEE/SouthParkData`
[3] `http://southpark.wikia.com/wiki/South_Park_Archives`

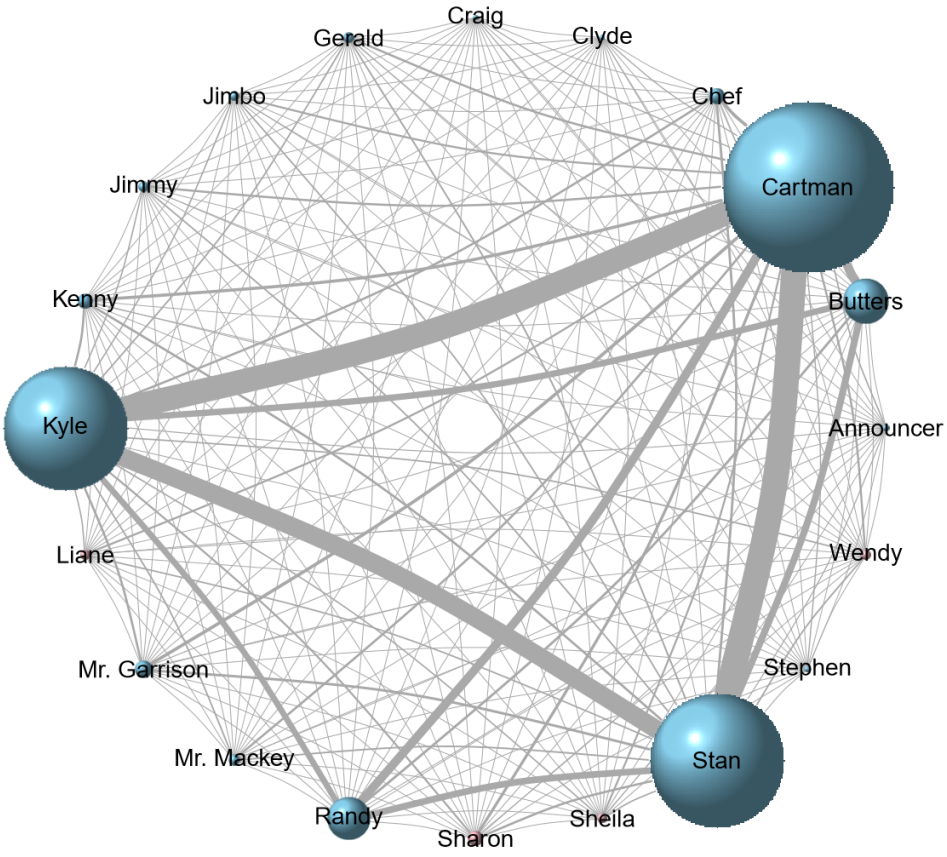

Figure 1: Co-occurences of characters over 257 episodes, size of vertex in relation to amount of lines

4 Pre-processing

In general, the standard pre-processing tasks are executed on the corpus. Depending on the later use, different packages perform these. The package **tm** is used to convert the corpus to lower case, remove punctuation and numbers as well as strip whitespaces. Furthermore, stopwords are removed using the stopword list from the Snowball stemmer project. The corpus is stemmed using Porter's stemming algorithm.[4] It is used together with several DocumentTerm- and TermDocument-Matrices. From these DTMs/TDMs, sparse terms are removed. The maximum allowed sparsity depends on the data set and the intended use, with the allowed percentage being higher for the *by.episode* data set in comparison to *by.season* because there are more documents in the *by.episod* data set.

[4] `http://snowball.tartarus.org/algorithms/english/stop.txt`

Figure 2: Terms with highest TF-IDF score - *by.episode*

Figure 3: Terms with highest TF-IDF score - *by.season*

The pre-processing for the Structural Topic Model is done with the same tasks, however those are performed by the package **stm** itself.

For the Named Entity Recognition with package **openNLP**, no pre-processing tasks are performed. They can be executed and might improve the overall runtime, but it does not change the overall result. Especially stemming and the lower case convertion could decrease the results because stemmed or lower case entities, as well as the sentiments that are mined, will not be recognized.

5 Results and visualization

To sufficiently present the results and the corresponding visualization, each relevant script will present them in their own subsection.

5.1 Wordclouds

To start to gain an insight into South Park, I started with building wordclouds that display the most frequent words to see what South Park is about. However, even with stopwords removed, the words that are most frequent include words like *get, just* and *now*. From these words, nothing about the TV show can be derived, so to get meaningful answers to what is important in and characterisic for South Park, the corpus is built with a term frequency-inverse document frequency (tf-idf) weight, which reflects how important each term is to the corresponding document in the corpus. With this weighting, meaningful results can be derived, as can be seen in Figures 2 and 3.

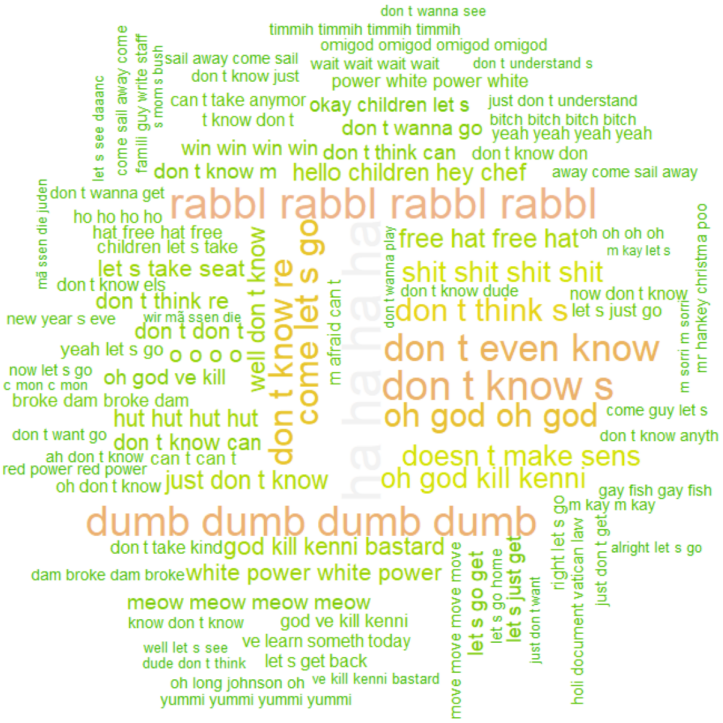

Figure 4: Wordcloud containing ngrams (n=4) by frequency

6 Conclusion

In conclusion,

7 Future work