A Brief Introduction to Natural Language Processing

Tian(Simon) Yun
Advisor: Dr. Nicole Dalzell

11/23/2019

Abstract

How can we deal with text data? How can we tell whether a person is happy or not by what he says? Is it possible to classify the articles by modeling processes? To finish these tasks, we need to study about natural language processing (NLP). NLP is a combination of Statistics, Computer Science, and Linguistics. It allows us to preprocess text data and fitting models on text data. This paper will explore the basic concepts in text analysis, simple approaches in sentiment analysis, and concepts of topic modeling. Apart from the theories in these topics, we will also implement those techniques onto literature works and Twitter comment dataset to analyze the hidden sentiments, and to extract the latent topics behind thousands of words.

Contents

Introduction & Motivation

Among all the animals, only humans have the ability of talking. This ability is based on the invention of language, which is indispensable for human interactions. With words and sentences, we can express our ideas and feelings to others explicitly. However, it is not a simple task to evaluate text-related data, because of its size and ambiguity. As for the size, since we interact so often every day, the size of text-related data keeps expanding very quickly. Countless comments are created on social platforms, and hundreds of books and articles are published. For the ambiguity, one sentence can have disparate meanings under different scenarios. You can say "you play really well" to describe a player who perform amazingly, or, to make fun of a player who perform awfully. Therefore, we need to develop a way to uncover the true thoughts beneath people's words.

How can we deal with large amount of text data and the ambiguity of language? In order to better analyze and understand these data, natural language processing (NLP) is developed. This is an interdisciplinary field combining Statistics, Computer Science and Linguistics. With NLP, we give computers the ability to "understand" the way we communicate. For instance, computers can identify whether a review on a specific mobile phone is positive or not. They can classify articles into various topics. What is more, they can finish tasks of translation from different languages to the others extremely fast.

This paper is divided into four sections. Section 1 will introduce about fundamental techniques and core concepts in text analysis. Section 2 contains typical ideas and methods in sentiment analysis. In Section 3, we will discuss *latent Dirichlet allocation (LDA)* in topic modeling from a Bayesian statistics perspective. Section 4 is a conclusion of the paper and a brief introduction of future work.

Section 1. Fundamental Techniques in Text Analysis

Different from numeric datasets we usually deal with, text data contains sentences and words. The first big problem we need to solve is: How can we convert text data into well-defined fixed-length data in order to feed the data into our models? This conversion is defined as feature extraction or feature encoding. In this section, we will cover fundamental techniques for text data preprocessing and core concepts in text analysis: *removal of stop words*, *bag-of-words*, *term frequency and inverse document frequency*, and *N-grams*.

Before we start introducing these contents, several frequently used terms need to be defined. Say we have ten books written by Jane Austen. These ten books are known as a *corpus*, which is a collection of written texts. A *document* is a subgroup of the corpus. Thus, it can be either a book or just one sentence. We aim at converting each document into a format that is ideal for the following analysis or modeling process. Normally, we will tokenize the documents into lots of rows of data. In each row, there is a pair of word and word's property, which can be number of occurrences.

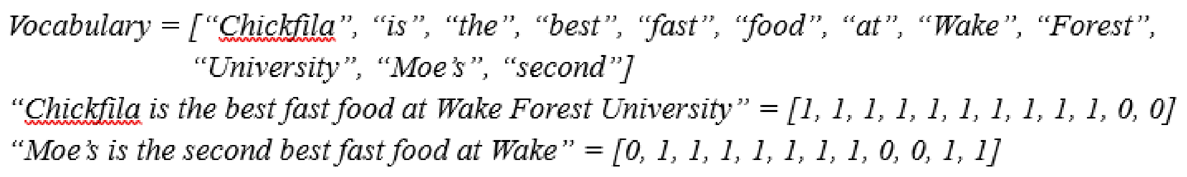
Section 1.1. Bag-of-Words

Bag-of-Words (BOW) is a stragihtforward method used to extract the features from text data for the modeling process by converting the data to vector representations. The key idea of BOW is that documents with similar contents are similar. Based on this idea, BOW classifies texts by a measure of the presence of known words. Thus, this method ignores the order or structure of words and only concentrate in whether known words show up in the document or not.

The known words are called *vocabulary*. Every word in the vocabulary should be unique, if we ignore the case and punctuation. One thing that is worth noticing is that only known words can be evaluated, while the unknown ones might simply be ignored. Therefore, the vocabulary needs to be carefully designed. If the vocabulary is too small, then lots of meaningful words will be omitted. On the contrary, if it is too large, then the vector representations of documents will be large and sparse. Such vector representations take up a large amount of memory and computational resources, and thus slower down the modeling process. In order to get rid of this problem, there are two main solutions: decreasing the size of vocabulary or creating a vocabulary of grouped words. To decrease the size of vocabulary, we can ignore the case and punctuation. Plus, we can adopt a method called *removal of stop words*. Stop words are the frequent meaningless words (*e.g.* the, a, is). What is more, the misspelled words should be corrected and some words can be reduced to their stem (*e.g.* "playing" to "play"). These are the basic data cleaning methods for text data. As for the creation of a vocabulary of grouped words, we actually combined every N consecutive words together and consider the combined entities as lots of single entities. This is known as *N-gram*, which will be introduced in detail in Section 1.3.

After the design of vocabulary, we need to measure the words according to different approaches: *binary scores*, *occurrences*, *frequencies*, and *TF-IDF*. Binary scores measures whether the words show up or not in a document by labeling the words with "1" or "0", where "1" indicates "do show up" and "0" indicates "not show up". Occurrences measure the number of times the known words show up in a document. In Figure 1, we consider each sentence as a separate document, and the lists are vector representations of the documents. The numbers in the lists are the occurrences of the corresponding words. Frequency measure computes the frequencies that each word appears in a document out of all the words in the document ($i.e.\ Frequency = \frac{Count(Number\ of\ Occurence\ of\ Word\ \text{“}any\ word\text{”})}{Count(Number\ of\ Unique\ Words)}$). TF-IDF is a measure that takes the significant rare words into considerations. It will be further explained in Section 1.2.

From above, we can see that bag-of-words method has some limitations. Firstly, it highly depends on the quality of vocabulary. The size of vocabulary is closely related with the sparsity of the vector representations of the documents. High sparsity of vector representations leads to high space and time complexity during the modeling process. Plus, it will be hard for the models to extract many meaningful results from them. Also, even if there is a nicely-designed vocabulary, the vector representations usually contain lots of zeros. Last but not the least, the order or the structure of the documents is ignored, but the order or structure might lead to disparate meanings of the words. For example, "this is interesting" and "is this interesting" represents completely different meanings in the documents.



Vocabulary = ["Chickfila", "is", "the", "best", "fast", "food", "at", "Wake", "Forest", "University", "Moe's", "second"]
"Chickfila is the best fast food at Wake Forest University" = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0]
"Moe's is the second best fast food at Wake" = [0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1]

Figure 1: Example of Bag-of-words with occurence measure

Section 1.2. Term Frequency & Inverse Document Frequency

Based on our intuition, it makes sense that the words which show up more often should be more significant in text analysis. Is it always the case? Not really. From what we mentioned before, stop words are such typical words that occur highly frequently but contain very limited information. Instead, the rare words among the documents might be more significant to distinguish the documents. In order to extract these rare and meaningful words, we need the technique named *term frequency – inverse document frequency (TF-IDF)*. TF-IDF is a measure that takes the meaningful rare words into account, and thus lowers the importance of the meaningless words with high frequency. *Term frequency (TF)* is a scoring of the frequency of the word in the current document. The larger the number of occurrences of a word within a document, the higher the value of TF. It is worth noticing that each document has different TF from others. Denoted $f_{t,d}$ as the number of occurrences of word t in document d and $t' \in d$ as word t' in document d, we have the following math notation,

$$TF_{t,d} = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}. \tag{1}$$

Inverse document frequency (IDF) is a scoring of how rare the word is across all documents. Such scores will highlight the words that are distinct in the given document. It means that a rare term has high value of IDF, while a frequent term has low value of IDF. IDF scores compute the weight of rare words across all documents in the corpus. Say N as number of documents, and n_t as number of documents where word t appears, its math notation is as below,

$$IDF = log\left(\frac{N}{n_t}\right). \tag{2}$$

In order to compute TF-IDF, we simply combine Equation 1 and Equation 2 together. Denoted $w_{t,d}$ as a TF-IDF score for a word t in a given document d, we have the following math notation,

$$w_{t,d} = TF_{t,d} \times IDF_{t,d} = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \times log\left(\frac{N}{n_t}\right). \tag{3}$$

$Word$	TF_A	TF_B	IDF	$TF-IDF_A$	$TF-IDF_B$
Chickfila	$\frac{1}{9}$	0	$log\left(\frac{2}{1}\right) = 0.3$	0.0333	0
Moe's	0	$\frac{1}{11}$	$log\left(\frac{2}{1}\right) = 0.3$	0	0.0272
is	$\frac{1}{9}$	$\frac{1}{11}$	$log\left(\frac{2}{2}\right) = 0$	0	0
the	$\frac{1}{9}$	$\frac{1}{11}$	$log\left(\frac{2}{2}\right) = 0$	0	0
best	$\frac{1}{9}$	$\frac{1}{11}$	$log\left(\frac{2}{2}\right) = 0$	0	0
second	0	$\frac{1}{11}$	$log\left(\frac{2}{1}\right) = 0.3$	0	0.0272
fast	$\frac{1}{9}$	$\frac{1}{11}$	$log\left(\frac{2}{2}\right) = 0$	0	0
food	$\frac{1}{9}$	$\frac{1}{11}$	$log\left(\frac{2}{2}\right) = 0$	0	0
at	$\frac{1}{9}$	$\frac{1}{11}$	$log\left(\frac{2}{2}\right) = 0$	0	0
Duke	$\frac{1}{9}$	0	$log\left(\frac{2}{1}\right) = 0.3$	0.0333	0
Wake	0	$\frac{1}{11}$	$log\left(\frac{2}{1}\right) = 0.3$	0	0.0272
Forest	0	$\frac{1}{11}$	$log\left(\frac{2}{1}\right) = 0.3$	0	0.0272
Universty	$\frac{1}{9}$	$\frac{1}{11}$	$log\left(\frac{2}{2}\right) = 0$	0	0

Table 1: Computations of TF-IDF scores of 2 sample documents

It would be better to see the effects of TF-IDF scores with examples. Assume we have the following two sentences, where each sentence is a separate document.

$$\begin{aligned} &\textit{Document } 1 : \text{“}Chickfila\ is\ the\ best\ fast\ food\ at\ Duke\ University.\text{”} \\ &\textit{Document } 2 : \text{“}Moe's\ is\ the\ second\ best\ fast\ food\ at\ Wake\ Forest\ University.\text{”} \end{aligned}$$

From Table 1, we can see that TF-IDF scores of common words are zero's, indicating that they are not distinct within our corpus. The words with non-zero TF-IDF scores are "Chickfila", "Moe's", "second", "Duke", "Wake", and "Forest". These words are more significant while show up less frequently across the documents. From this example, we can see how powerful TF-IDF is at extracting key words.

Section 1.3. N-grams

As we mentioned before, bag-of-words approach does not take the order or structure of the words into consideration. Since the order and structure are important to remove ambiguity and to better understand the contents, we need a method to cope with this. One solution is *N-grams*. An N-gram is a sequence of N words. By this notion, a 2-gram (or bigram) is a two-word sequence of words, such as "best fast", "fast food", "food at", "at Wake", and "Wake Forest". A 3-gram (or trigram) is a three-word sequence of words, such as "best fast food", "fast food at", "food at Wake", and "at Wake Forest". Which of these N-grams have we seen quite frequently? They might be "fast food", "Wake Forest", and "best fast food". In Figure 1, it is a bag-of-unigrams representation of documents. Normally, a bag-of-bigrams representation will outperform bag-of-unigrams.

Section 1.4. N-gram Model

With N-grams, we can assign probabilities to sentences and sequences of words. This is known as *N-gram model.* N-gram model predicts the occurrence of a sequence of words based on the occurrence of its previous (N - 1) words. When we use an N-gram model, we approximate the conditional probability of the next word. Denoted w_n as the current word, w_{n-1} as the previous (N - 1) words, and w_1^{n-1} as the sequence of words from the beginning of a document or corpus to the previous word of w_n, we have the following approximation

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1}).$$

This approximation is based on the assumption that the probability of the occurrence of a word depends only on the previous word. We say that N-gram model model is a type of Markov model which assumes that we can predict the probability of some future unit without looking too far in the past.

We can use one of the most intuitive ways, maximum likelihood estimation, to estimate the probability function. With the same notations, we have the following equation to compute the probability:

$$P(w_n|w_{n-1}) = \frac{Number\ of\ Occurrences\ of\ word\ w_n}{Number\ of\ Occurrences\ of\ w_{n-1}}.$$

Let's say we have the following corpus.

$$Document\ 1 : \text{“}He\ is\ a\ senior\ undergraduate\ at\ Wake\ Forest\ University.\text{”}$$
$$Document\ 2 : \text{“}I\ need\ to\ wake\ him\ up.\text{”}$$
$$Document\ 3 : \text{“}I\ usually\ wake\ up\ at\ nine\ in\ the\ morning.\text{”}$$
$$Document\ 4 : \text{“}It's\ hard\ to\ wake\ up\ early.\text{”}$$

We will use a bigram model (N = 2) and predict the occurrence of a word given only its previous (2 - 1 =) 1 word. We can compute the probability of occurrence of the word “up” after occurrence of the word “wake”:

$$\begin{aligned} P(up|wake) &= \frac{Number\ of\ Occurrences\ of\ word\ \text{“}wake\ up\text{”}}{Number\ of\ Occurrences\ of\ word\ \text{“}wake\text{”}} \\ &= \frac{2}{4} \\ &= 0.5 \end{aligned}$$

P(up | wake) = 0.5 implies that, within this corpus, the probability of seeing word “up” following the occurrence of the word “wake” is 50%. Similarly, we can compute P(forest | wake) = 0.25 and P(him | wake) = 0.25. This means that if we create a next word prediction based on our corpus, and a user types “wake”, we will give three options: “up”, “forest”, “him”

	i	need	to	wake	him	up	it's	hard	early
i	0	1	0	0	0	0	0	0	0
need	0	0	1	0	0	0	0	0	0
to	0	0	0	2	0	0	0	0	0
wake	0	0	0	0	1	1	0	0	0
him	0	0	0	0	0	1	0	0	0
up	0	0	0	0	0	0	0	0	1
it's	0	0	0	0	0	0	0	1	0
hard	0	0	1	0	0	0	0	0	0
early	0	0	0	0	0	0	0	0	0

Table 2: Bigram counts matrix for Document 2 and Document 4. In the second row with word "I", the value of 1 indicates that the count of occurrences of "need" after "I" is 1.

in such an order.Thus, we can see that N-gram models are useful at next word prediction. It is also useful at spelling check and correction. For example, in our corpus, if someone types "wake upp", we can correct it to "wake up", because we know that the probability of occurrence of the word "up" after word "wake" is high, and that "upp" overlaps with "up" much.

There are some limitations for N-gram models. Firstly, it is not feasible to train the model using the whole corpus, especially when the size is large. However, if we train N-gram models with limited size of corpus, then some possible sequences of words might be omitted. Secondly, the performance of N-gram models are highly dependent on the quality of training corpus. Specifically, there cannot be spelling errors in the corpus. Otherwise, the predictions of words might be inaccurate. Thirdly, an N-gram matrix (Table 2) for any training corpus is very likely to be sparse. Again, the sparse data is not ideal for modeling processes, since it will use up computational resources.

Section 1.5. Spam Filter & Word Hashing

One of the most annoying thing in our email inbox is spam email. However, with the development of NLP, the occurrences of spam emails is getting fewer and fewer. In Section 1.1, we talk about bag-of-words (BOW) representation. The key idea of BOW is that documents with similar content are similar. Therefore, if we regard every email as a separate document, we can convert every email into a BOW vector. Supposed we have the following two emails:

Document 1 : "*click the link to win ten thousand dollars per week.*" (*spam*)

Document 2 : "*i would like to discuss some questions on homework.*" (*not spam*)

Say that we have a vocabulary with 18 words, the BOW representations of our documents is as below:

$$\begin{aligned}Vocabulary &= [\textit{click},\ \textit{the},\ \textit{link},\ \textit{to},\ \textit{win},\ \textit{ten},\ \textit{thousand},\ \textit{dollars},\ \textit{per},\ \textit{week},\ \textit{i},\ \textit{would},\\ &\quad \textit{like},\ \textit{discuss},\ \textit{some},\ \textit{questions},\ \textit{on},\ \textit{homework}]\\ Document\ 1 &= [1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0]\\ Document\ 2 &= [0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1]\end{aligned}$$

Since the first document is a spam email, words, like "click", "link", "win", will be labeled as words that a spam might use. With these BOW vectors, we can train a binary classifier, such as logistic regression, classification trees, and so on. If we want to check whether a new email is spam or not, we just need to convert the email into BOW vector, feed this vector in the model, and gain the prediction.

Is the issue solved? Definitely not. People who create the spam are shrewd. They can modify the words in their emails so that the words are unknown to the models, but are still readable for people. For example, a modified version of spam email in our example can be:

$$\begin{aligned}Document\ 3 &: \text{``}\textit{i would like to discuss some questions on homework. c1ick teh 1ink 2 w1n}\\ &\quad \textit{tenn th0usand d0llars perr weeek.}\text{''}\\ BOW\ vector &= [0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1]\end{aligned}$$

What is the issue with this? It is that because the modified words, such as "click" to "c1ick", are not in our vocabulary, then BOW method will not count those words. Such an issue is known as *filter circumvent* or *out-of-vocabulary issue*. This leads to a BOW vector that is the same as the BOW vector of Document 2. Therefore, our model will classify this email as a non-spam normal email. The first solution that might come up in our mind is to add these new spam words into our vocabulary. However, this solution is not a wise one, since it will modify the size of our vocabulary, and thus lengthen the BOW vectors of the documents. Also, because we need to train the model with fixed-length training examples, we might have to retrain our model if the size of vocabulary is modified. What is more, before we finish fitting the new model, the old model will keep labeling the spam emails as non-spam ones. In all, we need a new solution which can cope with filter circumvent issue and does not require us to retrain the models when there is any new word.

The solution is *hashing*, which is an idea in computer science. We will use a designed hash function to map the words to a positive integer, for example "a" might be mapped to 1, and "b" is mapped to 2. Hash functions have the following characteristics:

1. With the same word, a hash function will always return the same value.
2. Hash function must determine the range of possible outputs.
3. Hash function is one-way, which means that we cannot determine the input from output.
4. Hash function might return same values for different input values, which is known as *collision*.

How can we use the concept of hashing to solve the out-of-vocabulary issue? We will combine hashing with BOW. This process is known as *word hashing*, or *feature hashing*. To set up, we need to create a really large array, for example one with size of 2^{32}. The corresponding value for each index is called *feature value*, and it will be initialized to 0. Afterwards, a hash function needs to be carefully designed and will be used to map the words to integers between 0 and 2^{32}. Note that the output values do not include 2^{32}, since the array has the starting index of 0. We can now feed every word in a document into the hash function. Then, we use the output integers to refer to the indices in our array, and increment the feature values of these indices by 1. This final array will be our new BOW of the document. The new BOW vector for Document 2 will be:

$$New\ BOW\ for\ Document\ 2 : [0,\ 1,\ 1,\ ...,\ 1,\ 1,\ 0,\ ...,\ 1,\ 1,\ 1,\ 1,\ 0,\ 1]$$
$$(2^{32}\ feature\ values)$$

Word hashing can solve out-of-vocabulary issue, because every single word, whatever known or unknown, ends up with incrementing some feature values. Therefore, we do not need to retrain our model any more, since word hashing is able to react with new words quickly. This is also known as *online learning*. How can the model know that the word is spam-related or not? The model will learn this when a user classifies an email as spam, then the probabilities of words in that email to be spam-related will increase.

There are several issues with word hashing. This idea is highly dependent on the quality of hash function and the size of array. Both hash function and the size need to be nicely designed in order to get rid of collisions. Otherwise, a spam word might be mapped to the integer representing a non-spam word. Secondly, the accuracy of the model will be lower due to the new words. However, with prolonged learning periods, the model is able to handle with the new words. Thirdly, we can tell that the BOW vector generated from word hashing process is sparse, since the original array is very large. Last one also the most important one, the hash function is one-way. It means that, even though our model can work successfully, and we know which index is labeled as “spam word”, we fail to know what the spam word is. Thus, the interpretability of the model is limited.

Section 2. Sentiment Analysis

This section will introduce the applications of methods that are discussed in Section 1 in R. We will use `gutenbergr` and `janeaustenr` libraries to access text data about books and articles written by different authors, and also use the Twitter comment data from Kaggle [23] as our text data. Sentiment analysis over these datasets will be carried out, and R code and frequently used functions will be explained. Some important packages will be used are: `ggplot2`, `tidyverse`, `tidytext`, `tidyr`, `dplyr`.

Section 2.1. Tidy Text Format & Tokenization

According to Silge et al. [21], Tidy text format is defined as a table with one-token-per-row. Every token is a meaningful unit of text, such as a single word, an n-gram, or a sentence.

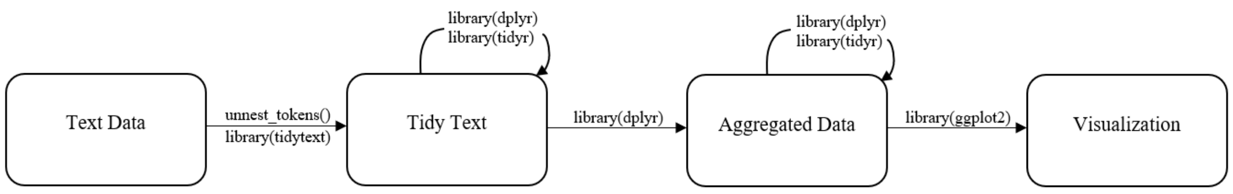


Figure 2: Flowchart of a typical text analysis using tidy text data

The process of converting documents into tokens is known as tokenization. The tidy text format is useful when we are using R to implement text analysis, since there are various tools we can use with, such as `ggplot2` and `tidyr`.

Figure 2 shows the typical process of implementation of text analysis using tidy text data. Text data is the original data stored as a list. Let the following code be the text data of our documents:

```r
text <- c("He is a senior undergraduate at Wake Forest University.",
          "I need to wake him up.",
          "I usually wake up at nine in the morning.",
          "It's hard to wake up early.")
```

We can use `tibble()` to convert text data into a data frame that is consistent with tidy tools. This function is available in `library(dplyr)`. Two columns are in `text_df`: the first column named `line` indicates the document number, the second column named `text` stores the data of interest.

```r
text_df <- tibble(line = 1:4, text = text)
tokenized_df <- text_df %>% unnest_tokens(word, text)
```

Can we use this tibble to start analyzing? Not yet. Because we need a table with one-token-per-row. If we want to convert tibble to a tidy data structure and tokenize the documents into words, we need to use `unnest_tokens()` in `library(tidytext)`. This function will tokenize the documents into words. There are two columns within `tokenized_df`: the first column named `line` indicates the documents the tokens belong to, the second column named `word` indicates the tokens. Therefore, since we have six words in the second document, there will be six rows of data relevant to Document 2 in `tokenized_df`. A good thing about `unnest_tokens()` is that it will ignore punctuation and also modify the words into lowercase. These will make the following analysis simpler.

Section 2.2. Necessity of Removal of Stop Words

Before we directly use the tokenized data, we need to go through the basic text data preprocess — removal of stop words. We have mentioned about why we need to remove the stop

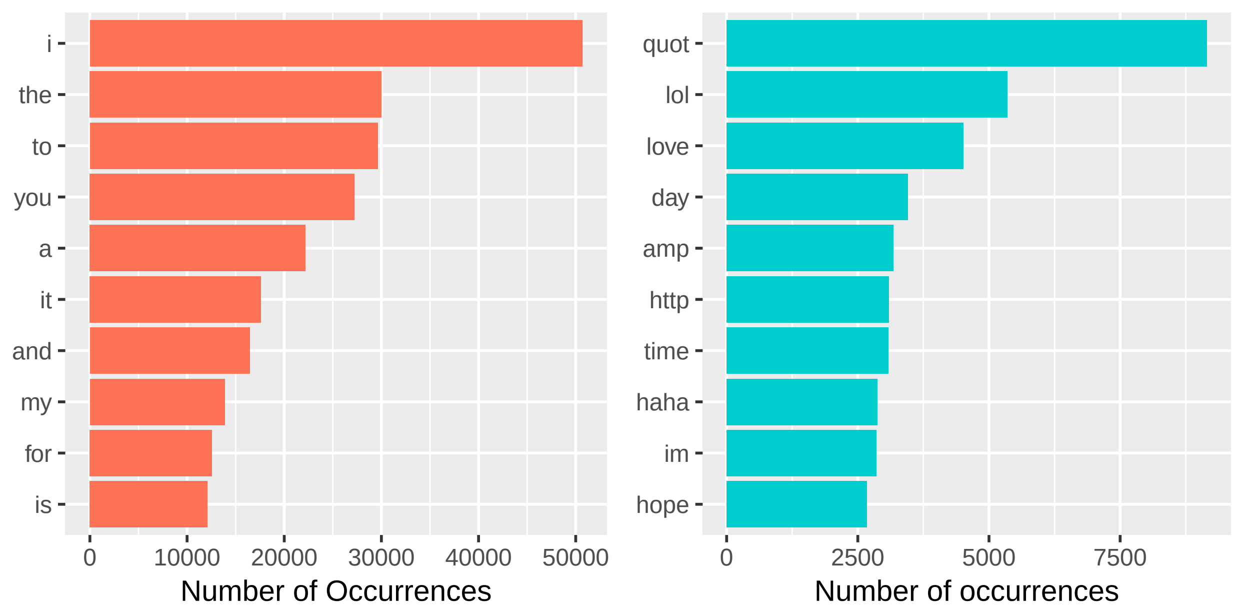


Figure 3: Necessity of removal of stop words. Left plot does not remove stop words, while right plot does remove stop words.

words. In this section, we will show why the removal of stop words is so necessary and how we can do it.

```r
data("stop_words")
clean_tidy_df <- text_df %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)
```

We will use Twitter comment dataset as an example. From the above code, if we want to remove the stop words, we need to load the built-in vocabulary of stop words, and use `anti_join()` to remove the stop words.

Figure 3 demonstrates the drastic influence to our following analysis if we do not remove the stop words. If we start the analysis without the removal, we need to deal with lots of meaningless words, like “I”, “the”, “to”. These are words that will show up highly frequently in people’s comments. On the stark contrast, after removal of stop words, the remaining words reflect that people enjoy using words, like “quot”, “lol”, “love”, in their Twitter comments. However, we can see one stop word “im”, because this word is not included in the built-in vocabulary of stop words. Therefore, we can modified the vocabulary of stop words by adding “im” in and use `anti_join()` to remove the stop words.

Section 2.3. Word Frequencies Analysis & Correlation Test

Now, we know how to tokenize our documents and preprocess the text data. Before the introduction of sentiment analysis, we will introduce word frequencies analysis and correla-

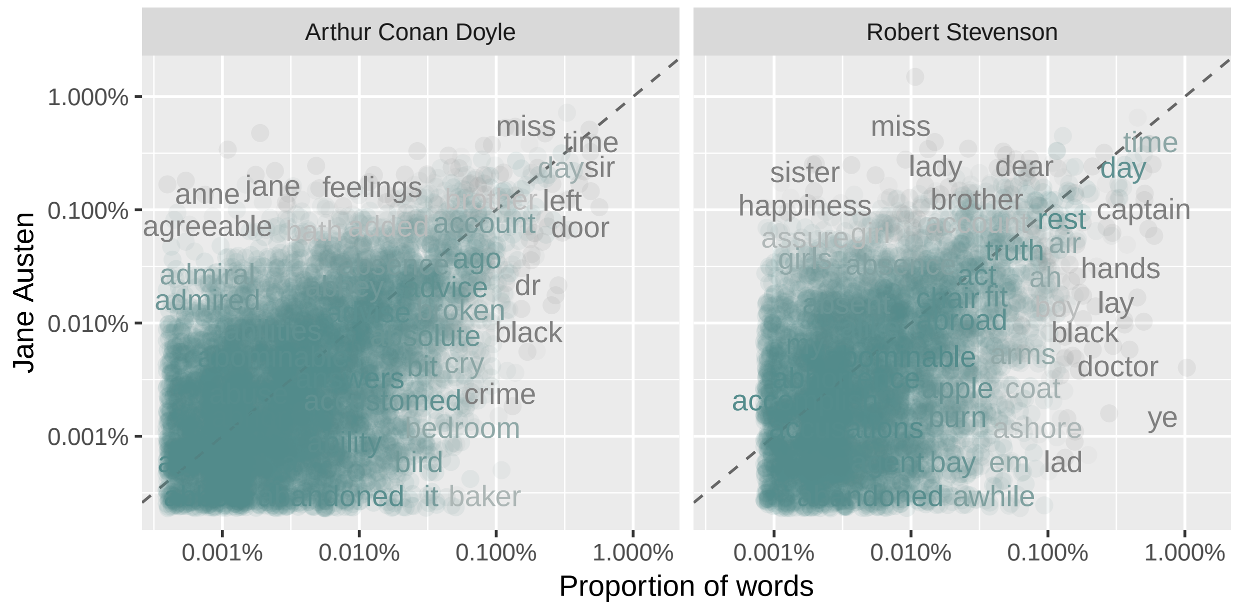


Figure 4: Word frequency analysis of books written by Jane Austen, Arthur Conan Doyle, and Robert Stevenson.

tion test. The data we will focus on is a collection of books written by Jane Austen, Arthur Conan Doyle, and Robert Stevenson.

In Figure 4, the color of a word is correlated to the difference between the frequency of this word in two authors' works. If the difference is greater, then the color of this word tends to be black. The words that are close to the line have the similar frequencies in the works from two authors. For example, words, such as "time", "miss", and "Sir", are used similarly frequently by Jane Austen and Arthur Conan Doyle. Words that are far from the line show up more frequently in one's work versus another's. For example, words, such as "ye" and "doctor", are used more often by Robert Stevenson versus Jane Austen.

If we compare Austen-Doyle plot with Austen-Stevenson plot, we can tell that low-frequency words in Austen's works might show up in Doyle's works, but not Stevenson's works. Plus, in Austen-Doyle plot, the words scatter less, and fewer words are black. These evidence reveals that Austen and Doyle tend to use similar words more often than Austen and Stevenson do. Also, there are fewer data points in Austen-Stevenson plot. How can we quantify the similarities of the words the authors would use? The answer is *correlation test*. We can carry out correlation test between Austen and Doyle by using the following R command:

```r
cor.test(data = frequency[frequency$author == "Arthur Conan Doyle",],
         ~ proportion + `Jane Austen`)
```

Based on Table 3, we can see that the words used by Austen are more correlated to those used by Doyle.

	t-statistics	df	p-value	correlation
Austen-Doyle	67.82	7468	<2.2e-16	0.617
Austen-Stevenson	42.83	5712	<2.2e-16	0.493

Table 3: Results of correlation tests between the authors.

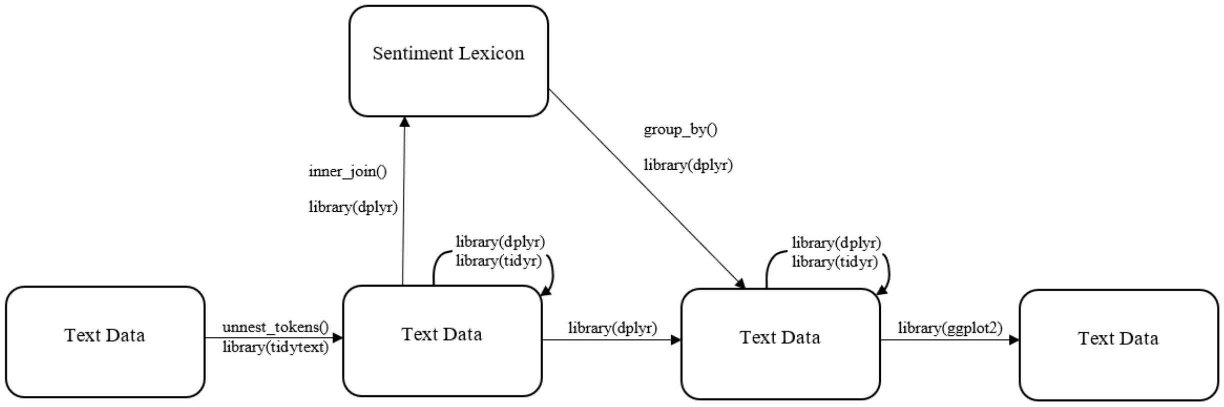


Figure 5: Flowchart of sentiment analysis with tidy text format.

Section 2.4. Sentiment Analysis on Literature Works

How can we quantify people's sentiment when they are writing books or articles? This section will introduce how to implement sentiment analysis with tidy text format. With sentiment analysis, we can disclose the hidden sentiments behind the words and sentences.

Section 2.4.1. Sentiment Trends & Most Common Positive/Negative Words

One common way to carry out sentiment analysis is to assign sentiment to every single word in a document, and then the sentiment of this document will be the sum of the sentiments. Thus, we need sentiment lexicons in order to know the sentiments of the words. In `library(tidytext)`, there are three sentiment lexicons: *AFINN*, *bing*, and *nrc*. These lexicons are based on unigrams, and they are not exactly the same. The AFINN lexicon assigns words with scores from -5 to 5, where negative values stand for negative sentiments and positive values stand for positive sentiments. The bing lexicon assigns words with "negative" or "positive". The nrc lexicon assigns words with several sentiments in a binary fashion. The sentiments are: positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust. To access either of these lexicons, we can use:

```
get_sentiments("afinn")
```

There are several concerns over these lexicons. One concern is that these are general sentiment lexicons. If we need to carry out sentiment analysis in a specific field and a field-specific sentiment lexicon is available, the field-specific one is preferred. Also, not all English words are in these lexicons, since there are lots of neutral words.

There are some issues with the approach of summing up the sentiments of single words within a document. First of all, the order or structure of words is not taken into consideration, such as “not good” or “not happy”. Also, the large size of a document might reduce the overall sentiment to neutral. Therefore, we need to be careful to pick an optimal size of documents if we adopt this approach.

Now, let’s implement this approach on literature works written by Arthur Conan Doyle. We will download the books from `library(gutenbergr)`, and use `mutate()` to create a new column that indicates the titles of the books.

```
conandoyle <- gutenberg_download(c(1661, 834))
conandoyle_book <- conandoyle %>%
  mutate(book = ifelse(gutenberg_id == '1661',
  'The Adventures of Sherlock Holmes',
  'The Memoirs of Sherlock Holmes'))
```

Afterwards, we will use `group_by()` and `mutate()` to assign row numbers for every line of contents in each book. The contents will be tokenized to one-unigram-per-row. `inner_join()` is used to assign the sentiment to each word by using bing sentiment lexicon. We will split each book into sections of 80 sentences by using `count()`. Therefore, our documents are 80-sentence chunks. At last, we use `spread()` to modify the dataset so that negative and positive sentiments have separate columns.

```
conandoyle_sentiment <- conandoyle_book %>%
  group_by(book) %>%
  mutate(linenumber = row_number()) %>%
  ungroup() %>%
  unnest_tokens(word, text) %>%
  inner_join(get_sentiments("bing")) %>%
  count(book, index = linenumber %/% 80, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
```

We can now use this processed dataset to plot the sentiment trends in each book (Figure 6). In *The Adventures of Sherlock Holmes*, the start of the book is relatively peaceful. This can be attributed to the fact that the start of the book is about how Sherlock Holmes met with John Watson. The middle part of the book is mostly negative. This might be caused by the murders, weird and dangerous cases, and the appearances of Professor Moriarty. We can also see that, at the end of the book, the sentiments fluctuate much.

Additionally, we can analyze what the most common positive and negative words are (Figure 7). Notice that the most common negative word “miss” can be the title for unmarried women. We can add this word in vocabulary of stop words to get rid of it.

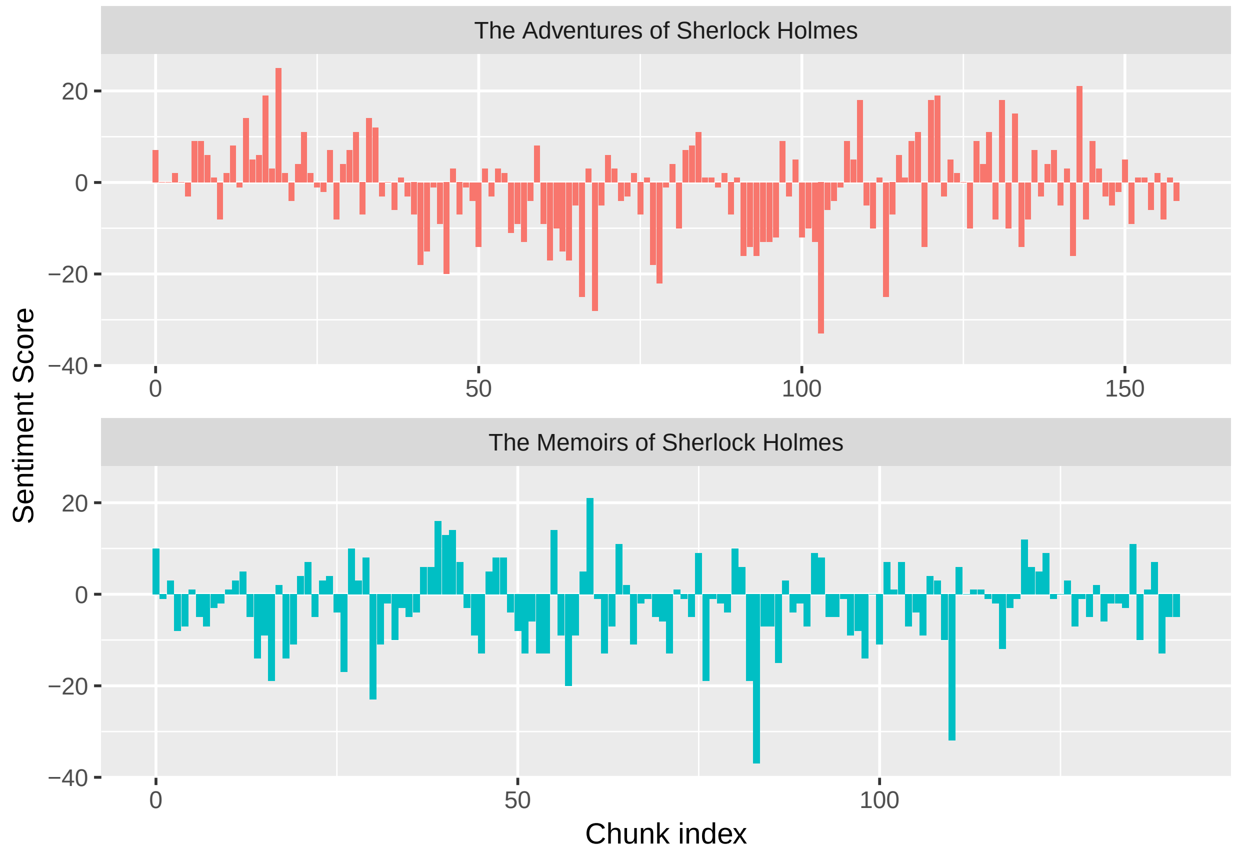


Figure 6: Sentiment trends in literature works written by Arthur Conan Doyle.

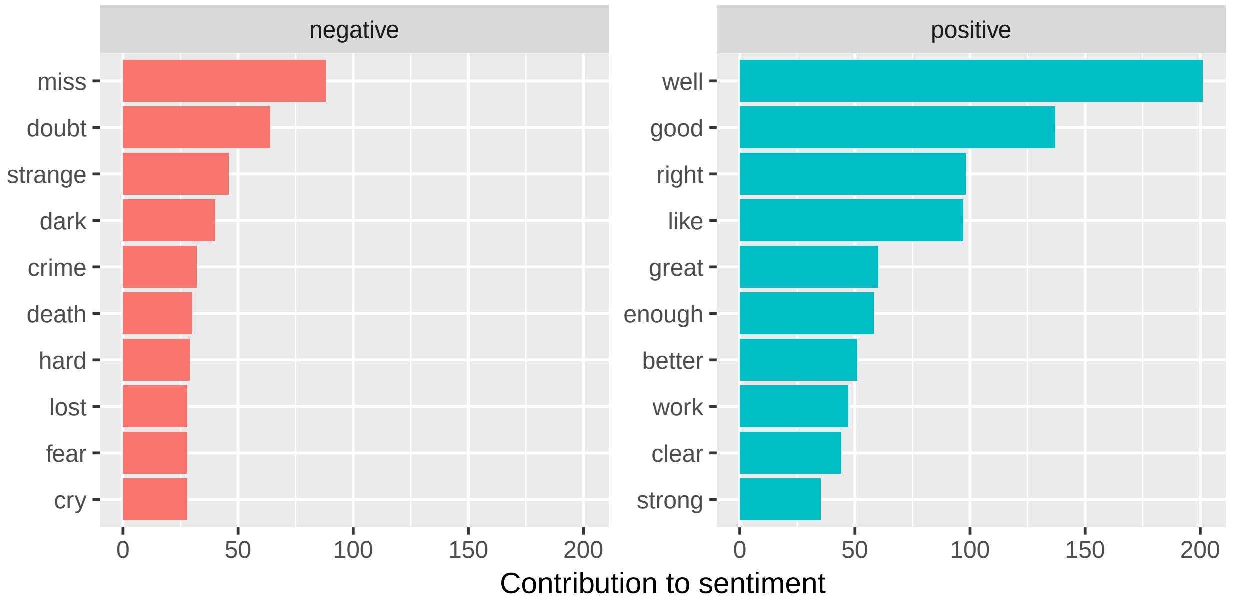


Figure 7: Most common positive and negative words in *The Adventures of Sherlock Holmes*

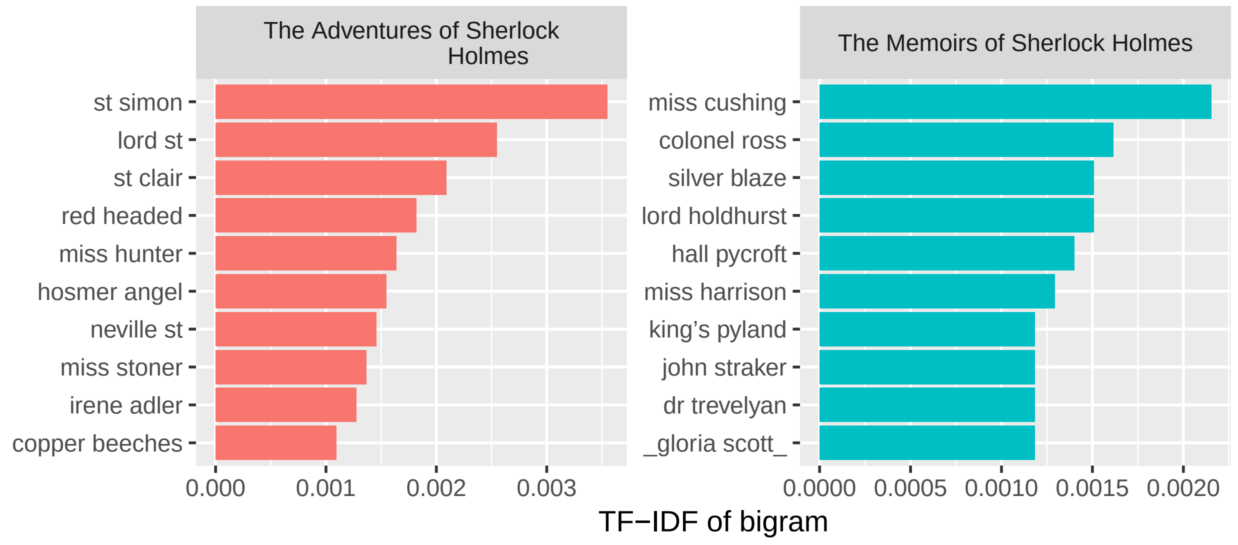


Figure 8: Bigrams with top 10 TF-IDF values in each literature works by Arthur C. Doyle.

Section 2.4.2. Extraction of Uncommon Meaningful Phrases

How can we find the uncommon but meaningful 2-word phrases in each book written by Doyle? We can combine the concepts of TF-IDF and bigrams. In order to tokenize the books to bigrams, we can use the following R command:

```
doyle_bigrams <- conandoyle_book %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2)
```

However, we cannot simply start computing the occurrences of bigrams, since the stop words have not been removed yet. Since the tokens are bigrams, but the stop words are unigrams. To get rid of bigrams containing stop words, we can use the following R code. we need to use `separate()` to split bigrams by delimiter " " (a space) so that we can have two columns representing the words in bigrams. We then use `filter()` to remove the bigrams containing stop words. Finally, in order to combine the words back together, we use `unite()`.

```
doyle_clean_united <- doyle_bigrams %>%
  separate(bigram, c('word1', 'word2'), sep = " ") %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word) %>%
  unite(bigram, word1, word2, sep = " ")
```

In order to compute TF-IDF values for each bigram, we can use `bind_tf_idf()` as below.

```
doyle_bigrams_count <- doyle_clean_united %>%
  count(book, bigram) %>%
  bind_tf_idf(bigram, book, n) %>%
  arrange(desc(tf_idf))
```

According to Figure 8, we can see that most of the uncommon but informational 2-word phrases are names of characters and places. These phrases might be able to distinguish one literature work from another, since they are unique to the corresponding novels.

Section 2.5. Sentiment Analysis on Twitter Comments

We have basic ideas about how to carry out sentiment analysis in R. In this section, we will introduce concepts about *pairwise correlation* and implement simple approach to carry out sentiment prediction on Twitter comment dataset.

Section 2.5.1. Pairwise Correlation

Correlation among words is used to examine how often the co-occurring words appear together relative to how often they appear separately. In this section, we will introduce *phi coefficient*, also known as *mean square contingency coefficient*, which is a common measure for correlations of binary variables. Two binary variables here are: the presence of word X, and the presence of the following word Y. The following table can help us to understand this concept.

	Has word Y	No word Y	Total
Has word X	n_{11}	n_{10}	$n_{1(\cdot)}$
Has word X	n_{01}	n_{00}	$n_{0(\cdot)}$
Total	$n_{(\cdot)1}$	$n_{(\cdot)0}$	n

In this table, n_{11} represents the number of documents where both word X and Y appear together, while n_{00} represents the number of documents where neither of words appear. Here, a document can be a sentence, a paragraph, or even a book. Using a sentence as a document is recommended, since the probability of presence of both words X and Y in a paragraph or a book can be very high. From above, the phi coefficient will be

$$\phi = \frac{n_{11}n_{00} - n_{10}n_{01}}{\sqrt{n_{1(\cdot)}n_{0(\cdot)}n_{(\cdot)1}n_{(\cdot)0}}}$$

We will split the data into sections of 100 comments. We can then use `pairwise_cor()` function in `library(widyr)` to compute the phi coefficients between words. It can be applied as below.

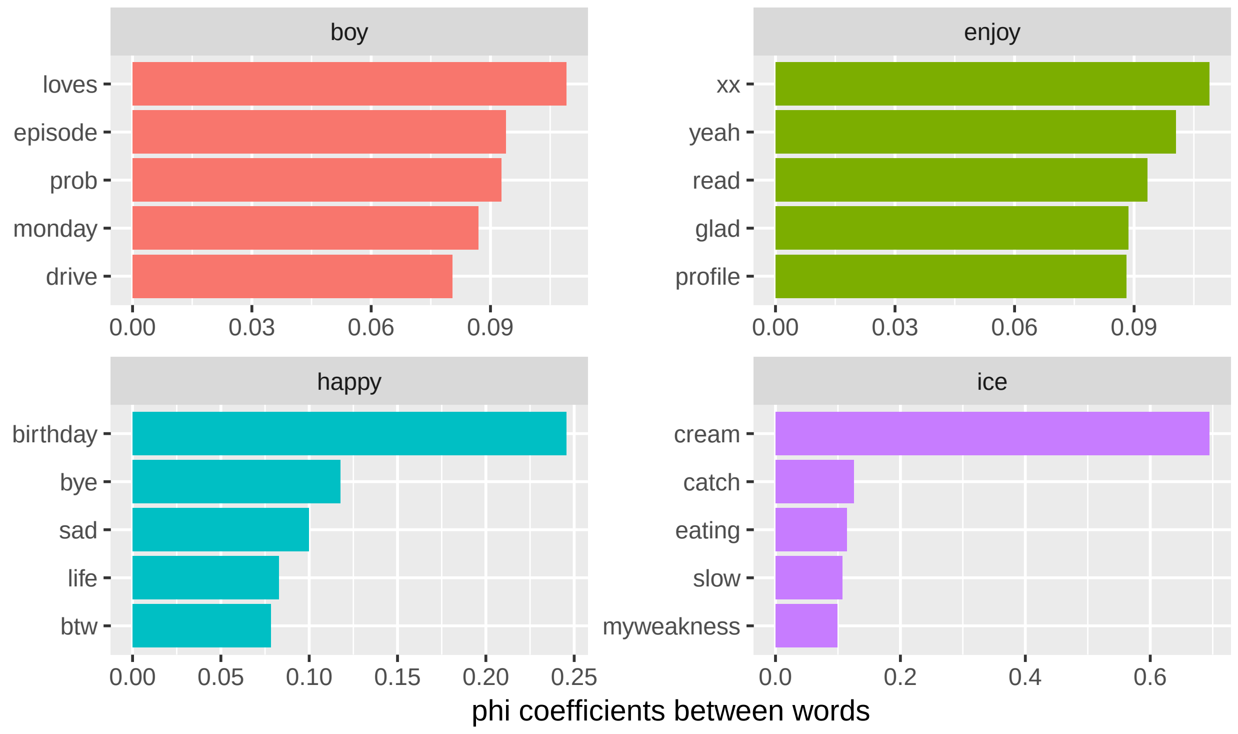


Figure 9: Pairwise correlation analysis of chosen words.

```
twitter_word_cors <- twitter_df %>%
  mutate(section = row_number() %/% 100) %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words) %>%
  group_by(word) %>%
  filter(n() >= 100) %>%
  pairwise_cor(word, section, sort = TRUE)
```

With the processed data, we can find the words which are correlated with the chosen words the most. In Figure 9, we focus on words "boy", "enjoy", "happy", and "ice". We can see that "happy" is correlated with "birthday" the most, while "ice" is correlated with "cream".

Section 2.5.2. Sentiment Prediction

In Section 4, we learn that we can sum up the sentiments of words in a document to compute the overall sentiment of the document. In order to predict the overall sentiment of a comment in Twitter comment data, we can apply this approach. This dataset includes 99989 observations, where each row represents one comment. Every row has information about the true sentiment label. After removing the stop words and joining with AFINN lexicon, there are 56595 observations left. The confusion matrix is shown below.

	Label Positive	Label Negative	Total
Predicted Positive	15791	6726	22517
Predicted Negative	9994	24084	34078
Total	25785	30810	56595

From the confusion matrix, we can see that the accuracy of this approach is $\frac{15791+24084}{56595} \approx 0.705 = 70.5\%$. Precision is $\frac{15791}{25785} \approx 0.612 = 61.2\%$. Sensitivity is $\frac{15791}{22517} \approx 0.701 = 70.1\%$. We can see that such a simple and easy-to-implement approach actually works pretty well. However, there are several concerns with this approach. First of all, we cannot predict the sentiment of a comment which does not include any word that is in AFINN lexicon. Also, we have not processed the weird words people intentionally type. For example, someone who strongly agrees with something might comment "goooood", which is actually "good". Therefore, we miss the information of these kinds of words. What is more, we have ignored the order of the words, since our approach is based on unigrams.

Section 3. Topic Modeling

This section will mainly introduce topic modeling from Bayesian statistics perspective. Topic modeling is the process of labeling a collection of documents with the unknown topics that can best describe them. These hidden topics are known as *latents*, and they can only emerge during the modeling process. We will focus on *latent Dirichlet allocation* (LDA). Briefly, the goal of LDA is to label the documents with a fixed set of imaginary topics, where each topic represents a set of words. If the words in a document are mostly captured by one or more imaginary topics, then this document will be labeled as these topics. According to Ganegedara [9], the big idea behind LDA is that each document can be described by a distribution of topics and each topic can be described by a distribution of words.

Section 3.1. Background Knowledge of LDA

Before we systematically talk about LDA, we need to go through some basic concepts about Bayesian statistics, multinomial distribution, and Dirichlet distribution. This section will introduce these ideas and prepare us for the derivation of LDA.

Section 3.1.1. Prior & Posterior

Models we have learned about during the class are related to *frequentists statistics*. According to Rathi [20], frequentist statistics assumes the probability of an event is the long-run frequency of random events in repeated trials. The common flaw of frequentist approach is that p-values and confidence intervals are highly dependent on the sample size. If the sample size is small, then p-values and confidence intervals will not be significant. Plus, confidence intervals are constants but not probability distributions. Therefore, the uncertainty is removed by computing estimates.

Rather than computing estimates, *Bayesian statistics* focuses on applying probabilities to statistical problems. This implies that the uncertainty will be reserved and refined by adjusting individual beliefs based on new evidence. *Prior belief distribution* and *Posterior belief distribution* are used to assign such probabilities.

According to Rathi [20], prior belief distribution is used to represent our strength on beliefs about the parameters based on the previous experience. Posterior belief distribution is derived from multiplication of likelihood function and prior belief distribution. Denoted Y as the data (or evidence), θ as a parameter we want to estimate, we will have

$$P(\theta \mid Y) = \frac{P(\theta \; and \; Y)}{P(Y)} = \frac{P(Y \mid \theta) \cdot P(\theta)}{P(Y)}. \tag{4}$$

In Equation 4, $P(\theta \mid Y)$ is the posterior distribution, $P(\theta)$ is the prior distribution, $P(Y \mid \theta)$ is the likelihood function, and $P(Y)$ is the marginal likelihood function. Posterior distribution means that given the data Y, what the probability of seeing the parameter θ is. We can see that posterior is based on likelihood and prior. Likelihood function means that given the parameter θ, what the probability of seeing the data Y is. Prior distribution is a guess of the probability density function of parameter θ.

Section 3.1.2. Conjugate Priors

Based on Cohen et al. [4], a prior family is conjugate to a likelihood if the posterior is also a member of the prior family. For example, if a variable X represents a random draw from a normal distribution with mean θ and a fixed, known variance σ^2. Therefore, we have the density of X for a point x:

$$P(x \mid \theta) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{1}{2}\left(\frac{x-\theta}{\sigma}\right)^2\right). \tag{5}$$

Say that θ is drawn from a prior family that also has normal distribution with mean μ and variance σ^2, which means we assume that the variance of prior is equivalent to that of the likelihood. Denoted α as the hyperparameter (μ, σ^2), our prior is

$$P(\theta \mid \alpha) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{1}{2}\left(\frac{\theta-\mu}{\sigma}\right)^2\right). \tag{6}$$

Based on Equation 4, we can rewrite the posterior and plug in Equations 5 and 6:

$$\begin{aligned} P(\theta \mid x,\ \alpha) &\propto P(x \mid \theta) P(\theta \mid \alpha) \\ &\propto exp\left(-\frac{\left(\theta - \frac{x+\mu}{2}\right)^2}{\sigma^2/2}\right). \end{aligned} \tag{7}$$

From Equation 7, we can see that our posterior follows a normal distribution with mean $\frac{x+\mu}{2}$ and variance $\frac{\sigma^2}{2}$. Therefore, we say that the family of prior distribution is conjugate to the likelihood that has a normal distribution with a fixed, known variance.

Section 3.1.3. Multinomial Distribution & Dirichlet Distribution

Multinomial distribution and *Dirichlet distribution* are really important distributions in NLP. A multinomial distribution has one set of parameter, where each element in this set represents the probability of falling into one category. A random value drawn from multinomial distribution indicates a specific category. Therefore, the returned value is categorical. Let J as the levels of an unordered categorical variable Y, n_i as the number of occurrences of category i, and $\mathbf{p}$ as the probabilities of falling into the categories. Multinomial distribution is defined as:

$$\begin{gathered} Y \sim Multinomial(\mathbf{p})\ or\ Cat(\mathbf{p}), \\ P(Y = (Y_1 = n_1,\ \dots,\ Y_J = n_J) \mid \mathbf{p}) = \frac{\Gamma(\sum_{i=1}^{J} n_i + 1)}{\prod_{i=1}^{J} \Gamma(n_i + 1)} \prod_{i=1}^{J} \pi_i^{n_i}. \end{gathered}$$

Dirichlet distribution is similar to multinomial distribution, except it returns the probabilities of falling into categories. Dirichlet distribution has one set of parameter, which will determine the shape of distribution. Let X be the distribution of categories and it follows a Dirichlet distribution with the set of parameter $\boldsymbol{\alpha}$, x_i as the probability of being in category i, and K as the levels of categories of X. Dirichlet distribution is defined as:

$$\begin{gathered} X \sim Dirichlet(\boldsymbol{\alpha}), \\ P(X = (X_1 = x_1,\ \dots,\ X_K = x_K) \mid \boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{K} x_i^{\alpha_i - 1}, \end{gathered}$$

where

$$\begin{aligned} &\alpha_i > 0\ and\ K \geq 2, \\ &\sum_{i=1}^{K} x_i = 1\ and\ x_i \in (0,\ 1), \\ &B(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{K} \alpha_i\right)}. \end{aligned}$$

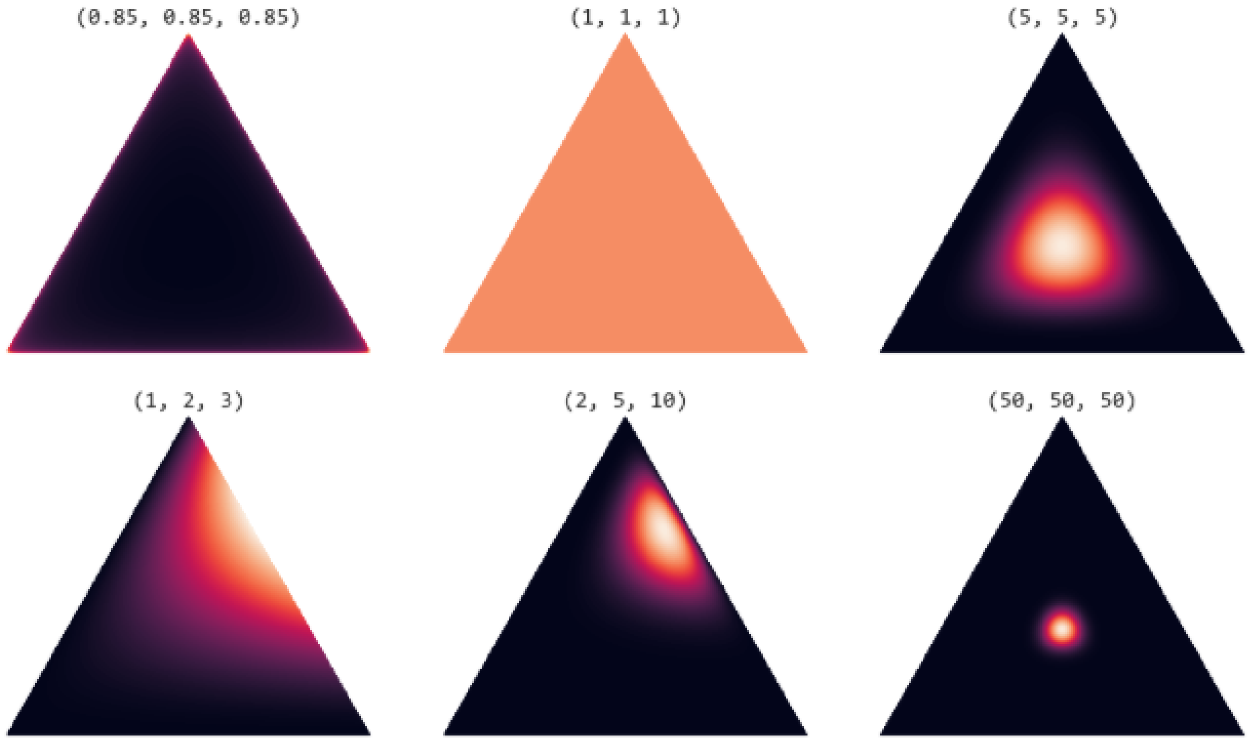


Figure 10: Dirichlet distribution on a 2-simplex for different values of α. Figure from Liu (2019).

The shape of Dirichlet distribution can be shown as $K-1$ simplex. Let α be this set of parameter. In Figure 10, the bright region indicates high values, while the dark region indicates low values. We can tell that the sum of elements in α determines how peaked the distribution is. For example, if $\alpha = [50,\ 50,\ 50]$, then there is a large value in the center of simplex. If $\alpha_i < 1$ for all i, then the peaks will show up at the corners.

Section 3.2. Latent Dirichlet Allocation (LDA) & Gibbs Sampling

Now, we are ready to talk about *latent Dirichlet allocation*. In Section 3, we have introduced that LDA is to label the documents with a fixed set of imaginary topics, where each topic represents a set of words. In order to simplify LDA, we assume that our corpus only contains one document. We need to define a list of variables as shown in Table 4. Since we assume there is only one document, then $M = 1$. The modeling process is shown in Figure 11. $\boldsymbol{w}$ is shaded, since it is the only observable variable. All the others are latent variables. Some random variables in LDA will follow multinomial distribution or Dirichlet distribution:

$$\boldsymbol{\theta} \sim Dirichlet(\boldsymbol{\alpha}) \tag{8}$$
$$\boldsymbol{\beta}_{k=1 \dots K} \sim Dirichlet(\boldsymbol{\eta}) \tag{9}$$
$$\boldsymbol{z}_{w=1 \dots N} \sim Multinomial(\boldsymbol{\theta}) \tag{10}$$
$$\boldsymbol{w}_{w=1 \dots N} \sim Multinomial(\boldsymbol{\beta}_{\boldsymbol{z}_w}) \tag{11}$$

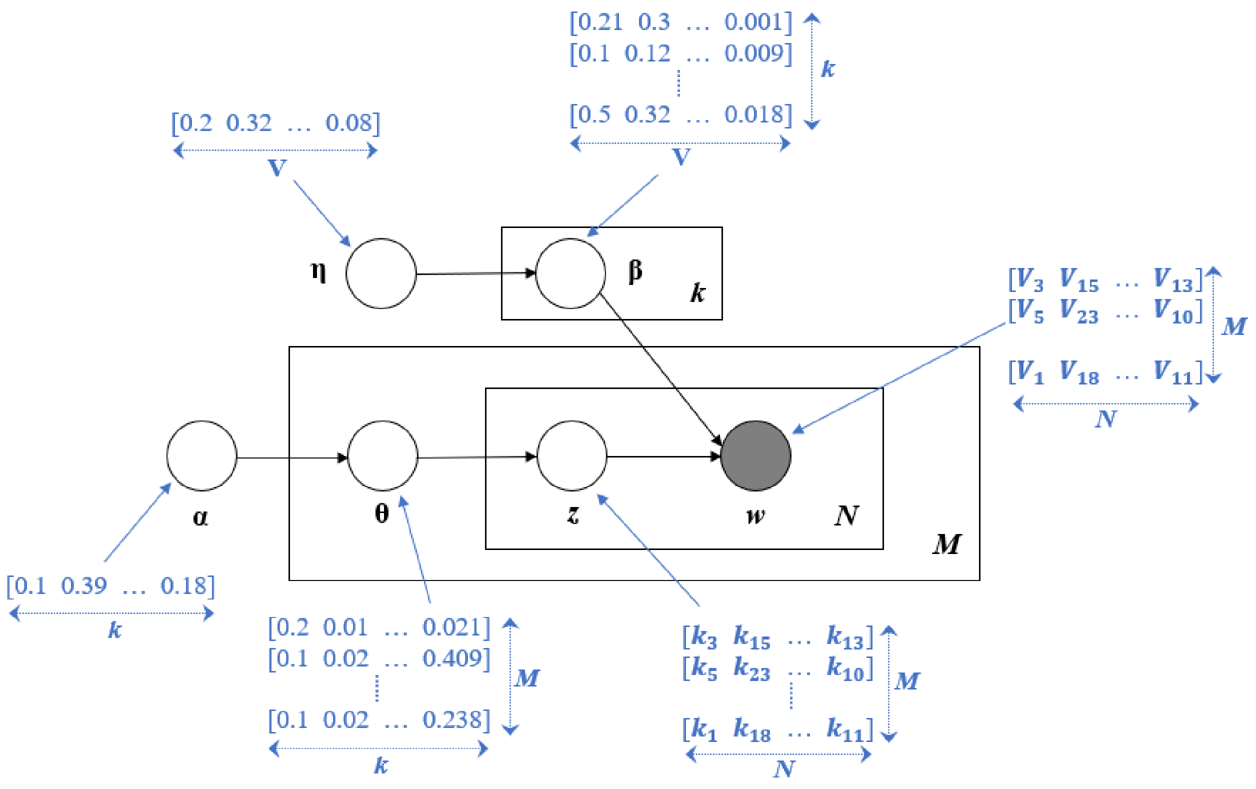


Figure 11: Graphical model of latent Dirichlet allocation. k_i in $\boldsymbol{z}$ indicates i^{th} topic, while V_i in $\boldsymbol{w}$ indicates j^{th} word in the vocabulary.

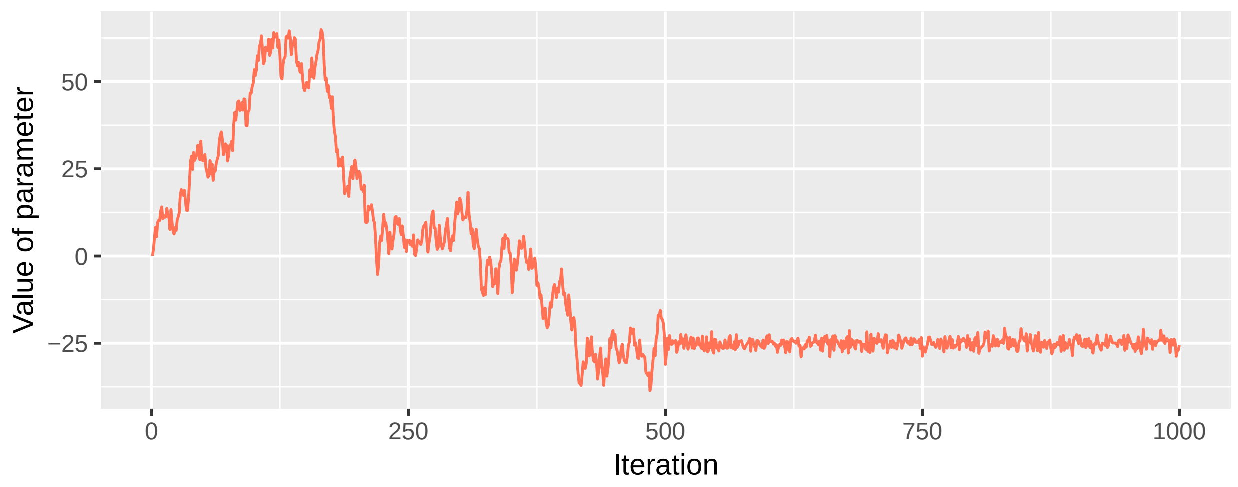


Figure 12: Sample traceplot of a parameter.

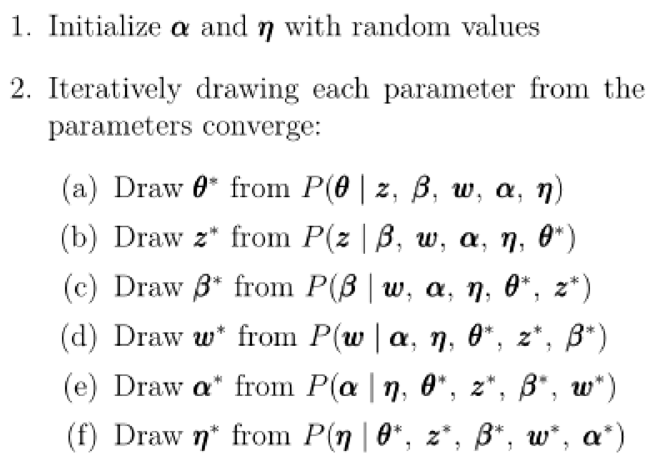
Variable	**Type**	**Meaning**
K	Integer	Number of possible topics
V	Integer	Size of vocabulary
M	Integer	Number of documents
$N_{d=1...M}$	Integer	Number of words in each document
N	Integer	Total number of words in the corpus
$w_{d=1...M,\ w=1...N_d}$	Integer between 1 and V	Identity of word w in document d
$\boldsymbol{\alpha}$	K-dimensional vector of positive values	hyperparameters of Dirichlet distribution of $\boldsymbol{\theta}$
$\boldsymbol{\eta}$	V-dimensional vector of positive values	Hyper parameters of Dirichlet distribution of $\boldsymbol{\beta}$
$\boldsymbol{\theta}$	$M \times K$ matrix	$\boldsymbol{\theta}_{ij}$ is the probability of i^{th} document to be classified as j^{th} *topic*
$\boldsymbol{\beta}$	$K \times V$ matrix	$\boldsymbol{\beta}_{ij}$ is the probability of i^{th} topic containing j^{th} word in the vocabulary
$\boldsymbol{z}$	$N \times K$ matrix	$\boldsymbol{z}_{ij}$ is the topic for j^{th} word in i^{th} document
$\boldsymbol{w}$	$N \times V$ matrix	Identity of all words in all documents

Table 4: Definition of variables in LDA.

These random variables are used to demonstrate the *generative process*, which is to generate a new document with N words. Equation 8 means that, with initialized hyperparameter $\boldsymbol{\alpha}$, we can compute $\boldsymbol{\theta}$, which is the word distribution of a topic. Equation 9 means that, with initialized hyperparameter $\boldsymbol{\eta}$, we can compute $\boldsymbol{\beta}_{k=1...K}$, which is the topic distribution of a document. Equation 10 implies that we will identify $\boldsymbol{z}_{w=1...N}$, which is the identity of topic of word w from a multinomial distribution with parameter $\boldsymbol{\theta}$. Equation 11 implies that given the topic $\boldsymbol{z}_w$ of word w, our generated word $\boldsymbol{w}_w$ is based on a multinomial distribution with parameter $\boldsymbol{\beta}_{\boldsymbol{z}_w}$. Therefore, we can use LDA to generate a new document with N words.

There comes two questions. Why do we need to generate new documents? Also, it seems that we can generate new documents even without an original dataset. Then, why do we need to feed the original dataset to LDA? These questions are closely related to the modeling process for LDA. We will introduce *Gibbs sampling method* for the modeling process of LDA. According to Cohen et al. [4], Gibbs sampling explores the state space, sampling explores the state space, sampling u_i each time, where p is the number of variables and $i \in \{1, ..., p\}$. These u_i are drawn from the full conditional distributions, $P(U_i \mid U_{-i},\ X)$, where U_{-i} denotes the set of $\{U_{i'} \mid i' \neq i\}$ and X denotes the data. In LDA, we have Gibbs sampling process as shown in Figure 13.

The sampling process will terminate when the parameters converge. One simple way to determine if a parameter has converged is to use *traceplot*. In Figure 12, we can see the parameter does not converge in about first 500 iterations. This period of iterations before convergence is called *burn-in period*. After the 500^{th} iteration, the parameter converges.



1. Initialize $\boldsymbol{\alpha}$ and $\boldsymbol{\eta}$ with random values
2. Iteratively drawing each parameter from the full conditional distributions until the parameters converge:
   (a) Draw $\boldsymbol{\theta}^*$ from $P(\boldsymbol{\theta} \mid \boldsymbol{z},\, \boldsymbol{\beta},\, \boldsymbol{w},\, \boldsymbol{\alpha},\, \boldsymbol{\eta})$
   (b) Draw $\boldsymbol{z}^*$ from $P(\boldsymbol{z} \mid \boldsymbol{\beta},\, \boldsymbol{w},\, \boldsymbol{\alpha},\, \boldsymbol{\eta},\, \boldsymbol{\theta}^*)$
   (c) Draw $\boldsymbol{\beta}^*$ from $P(\boldsymbol{\beta} \mid \boldsymbol{w},\, \boldsymbol{\alpha},\, \boldsymbol{\eta},\, \boldsymbol{\theta}^*,\, \boldsymbol{z}^*)$
   (d) Draw $\boldsymbol{w}^*$ from $P(\boldsymbol{w} \mid \boldsymbol{\alpha},\, \boldsymbol{\eta},\, \boldsymbol{\theta}^*,\, \boldsymbol{z}^*,\, \boldsymbol{\beta}^*)$
   (e) Draw $\boldsymbol{\alpha}^*$ from $P(\boldsymbol{\alpha} \mid \boldsymbol{\eta},\, \boldsymbol{\theta}^*,\, \boldsymbol{z}^*,\, \boldsymbol{\beta}^*,\, \boldsymbol{w}^*)$
   (f) Draw $\boldsymbol{\eta}^*$ from $P(\boldsymbol{\eta} \mid \boldsymbol{\theta}^*,\, \boldsymbol{z}^*,\, \boldsymbol{\beta}^*,\, \boldsymbol{w}^*,\, \boldsymbol{\alpha}^*)$

Figure 13: Gibbs sampling process in LDA.

Therefore, once the parameters in LDA converge, we can obtain the values of $\boldsymbol{\theta}$, $\boldsymbol{z}$, and $\boldsymbol{\beta}$. $\boldsymbol{\theta}$ can tell us the hidden topics of each document. $\boldsymbol{z}$ can tell us the identified topic of each word in each document. Since LDA model cannot give the name to a topic, we need to check $\boldsymbol{\beta}$ in order to name the hidden topic. For example, after modeling process, we observe $\boldsymbol{\theta}$ and know that a document $\mathbf{X}$ is labeled with topic k_3. If the words, such as "dog", "cat", and "elephant", are assigned with high probabilities of occurrences in $\boldsymbol{\beta_{k_3}}$, then we will name topic k_3 as "Animal". Afterwards, we can check $\boldsymbol{z}$ to see the set of words that actually fall into topic k_3 in this document $\mathbf{X}$. This is how we can use LDA model to classify the documents.

Section 3.3. Application of LDA on Twitter Comments

In this section, we will focus on Twitter comments dataset, and aim at classifying the sentiments of each comment by using LDA instead of the simple sentiment analysis approach mentioned in Section 2.4.

In R, `LDA()` is provided in `library(topicmodels)`. It is very convenient to use, but we need to convert our dataset from tidy text data to a document term matrix (DTM). A DTM has the following characteristics:

- each row represents one document,
- each column represents one term, such as a unigram, or bigram,
- each element represents the count of occurrences of the term in the document.

After we preprocess the dataset and inner join with `nrc` sentiment lexicon (*i.e.* mentioned in Section 2.4.1), we can use `cast_dtm()` to convert tidy text data to a DTM:

```r
twitter_tokens <- twitter_df %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words) %>%
  inner_join(get_sentiments('nrc')) %>%
  count(itemID, word)

twitter_dtm <- twitter_tokens %>%
  cast_dtm(itemID, word, n)
```

The parameters indicate the documents, terms, and number of occurrences respectively. After the preprocessing process, there are 60574 comments left. We can implement `LDA()` to classify the comments:

```r
twitter_lda <- LDA(twitter_dtm, k = 10, control = list(seed = 1123))
```

Here, `k = 10` means there are unknown 10 hidden topics, and `control` indicates a seed for the modeling process. The results are stored in `twitter_lda`. We can extract $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$:

```r
twitter_beta <- tidy(twitter_lda, matrix = 'beta')
twitter_theta <- tidy(twitter_lda, matrix = 'gamma')
```

Each row in `twitter_beta` has a topic, a term, and the probability of this term being in this topic. Each row in `twitter_theta` has a document, a topic, and the probability of this document being classified as this topic. Therefore, we can observe `twitter_theta` to know the topics of the documents. For example, we can use the following code to obtain the most possible topic of 61847^{th} comment and 80771^{th} comment:

```r
twitter_theta %>%
  filter(document == 61847) %>%
  top_n(1)
```

We can see that 61847^{th} comment is classified as Topic 4, while 80771^{th} comment is classified as Topic 8. Now, we can take a look at `twitter_beta` and check what Topic 4 and Topic 8 really are. Since `nrc` sentiment lexicon has 10 different categories, based on Figure 14, we can label Topic 4 as "Negative" and Topic 8 as "Joy". Now, let's check if our classification results match to the comments.

$$\begin{aligned} 61847^{th}\ Comment\ &: \text{“}@bigfatbees\ Haha,\ it\ was\ awful\ awful\ awful!\ But\ I'm\ determined \\ &\quad to\ get\ used\ to\ it.\text{”} \\ 80771^{th}\ Comment\ &: \text{“}@artvigil\ shot\ -12\ best\ ball\ 2\ man\ team\ over\ 2\ days.\ Got\ first \\ &\quad place\ for\ a\ nice\ stash\ of\ cash.\text{”} \end{aligned}$$

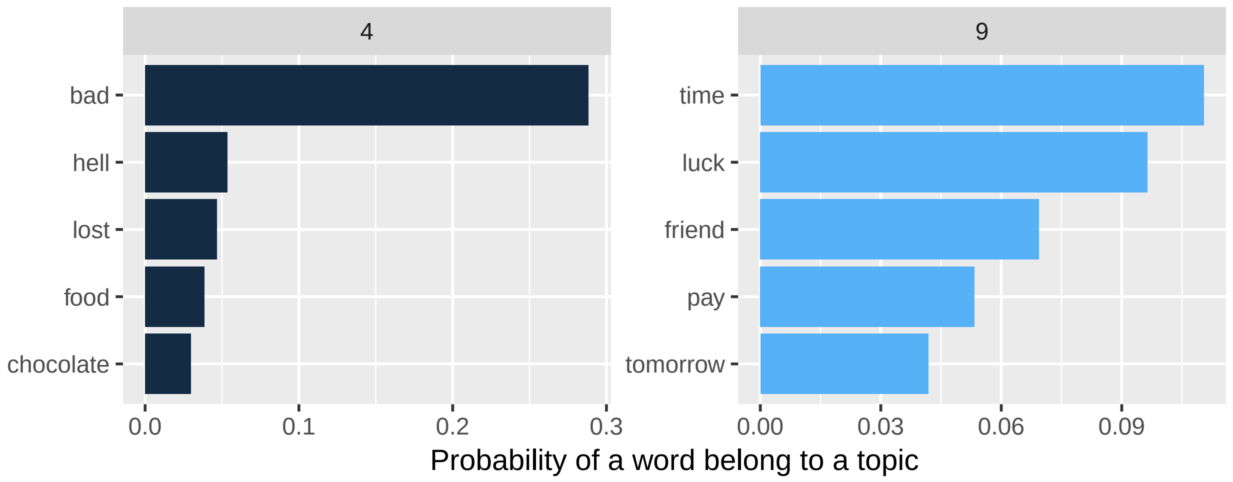


Figure 14: Distribution of words in Topic 4 and 9.

We can see that our results match to the comments! However, there are some issues. Since the size of every document is short, the results of LDA might be influenced. Also, we can see that “food” and “chocolate” might not be negative, but due to the high probabilities of “bad”, “hell”, “lost”, we still label Topic 4 as “negative”. This implies that the parameter `k` might be larger when we use `LDA()` command.

Section 4. Conclusion & Future Work

These are all the contents related to NLP to be discussed in this paper. We have covered the fundamental techniques and concepts in text analysis, including bag-of-words, TF-IDF, N-grams, N-gram model, and word hashing. Also, we go through the implementation of those techniques on literature works and Twitter comment dataset in R. Eventually, we briefly talk about topic modeling and provide a simple implementation in R.

However, there are still lots of materials that have not been covered in this paper. In the next semester, we aim at studying more in NLP. For LDA, we will introduce full conditional distributions, more derivations of LDA, collapsed Gibbs sampling method, variational Bayes. What is more, we might go through various methods in sentiment analysis and more preprocessing methods, such as stemming (*i.e.* "running" to "run"). With these materials, we are able to create cleaner dataset, generate more accurate results, and provide more comprehensive conclusions.

References: Coding

David Robinson (2019). gutenbergr: Download and Process Public Domain Works from Project Gutenberg. R package version 0.1.5. https://CRAN.R-project.org/package=gutenbergr

David Robinson (2019). widyr: Widen, Process, then Re-Tidy Data. R package version 0.1.2. https://CRAN.R-project.org/package=widyr

Grün B, Hornik K (2011). "topicmodels: An R Package for Fitting Topic Models." *Journal of Statistical Software*, *40*(13), 1-30. doi: 10.18637/jss.v040.i13 (URL: https://doi.org/10.18637/jss.v040.i13).

H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2016.

Hadley Wickham (2017). tidyverse: Easily Install and Load the 'Tidyverse'. R package version 1.2.1. https://CRAN.R-project.org/package=tidyverse

Hadley Wickham and Lionel Henry (2019). tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions. R package version 0.8.3. https://CRAN.R-project.org/package=tidyr

Hadley Wickham, Romain François, Lionel Henry and Kirill Müller (2019). dplyr: A Grammar of Data Manipulation. R package version 0.8.0.1. https://CRAN.R-project.org/package=dplyr

Julia Silge (2017). janeaustenr: Jane Austen's Complete Novels. R package version 0.1.5. https://CRAN.R-project.org/package=janeaustenr

Silge J, Robinson D (2016). "tidytext: Text Mining and Analysis Using Tidy Data Principles in R." *JOSS*, *1*(3). doi: 10.21105/joss.00037 (URL: https://doi.org/10.21105/joss.00037), <URL: http://dx.doi.org/10.21105/joss.00037>.

References

[1] Bayesian Statistics: A Beginner's Guide. QuantStart. https://www.quantstart.com/articles/Bayesian-Statistics-A-Beginners-Guide. Accessed 19 November 2019.

[2] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet Allocation. Journal of Machine Learning Research, 3, 993–1022. http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf. Accessed 19 November 2019.

[3] Brownlee, J. (2019, August 7). A Gentle Introduction to the Bag-of-Words Model. Machine Learning Mastery. https://machinelearningmastery.com/gentle-introduction-bag-words-model/. Accessed 19 November 2019.

[4] Cohen, S., & Hirst, G. (2019). Bayesian analysis in natural language processing. San Rafael, CA: Morgan & Claypool.

[5] Doll, T. (2019, March 11). LDA Topic Modeling. Medium. Towards Data Science. https://towardsdatascience.com/lda-topic-modeling-an-explanation-e184c90aadcd. Accessed 20 November 2019.

[6] Dirichlet distribution. (2019, November 4). Wikipedia. Wikimedia Foundation. https://en.wikipedia.org/wiki/Dirichlet_distribution. Accessed 19 November 2019.

[7] Dirichlet-multinomial distribution. (2019, November 4). Wikipedia. Wikimedia Foundation. https://en.wikipedia.org/wiki/Dirichlet-multinomial_distribution#For_a_set_of_individual_outcomes. Accessed 19 November 2019.

[8] Latent Dirichlet allocation. (2019, October 14). Wikipedia. Wikimedia Foundation. https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation. Accessed 19 November 2019.

[9] Ganegedara, T. (2019, March 27). Intuitive Guide to Latent Dirichlet Allocation. Medium. Towards Data Science. https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-latent-dirichlet-allocation-437c81220158. Accessed 19 November 2019.

[10] Geigle, C. (2016, October 15). Inference Methods of Latent Dirichlet Allocation. Universityof Illinois at Urbana-Champaign. http://times.cs.uiuc.edu/course/598f16/notes/lda-survey.pdf. Accessed 19 November 2019.

[11] Gibbs sampling. (2019, November 23). Wikipedia. Wikimedia Foundation. https://en.wikipedia.org/wiki/Gibbs_sampling#Mathematical_background. Accessed 25 November 2019.

[12] Kapadia, S. (2019, August 19). Language Models: N-Gram. Medium. Towards Data Science. https://towardsdatascience.com/introduction-to-language-models-n-gram-e323081503d9. Accessed 19 November 2019.

[13] Kristiadi, A. Gibbs Sampler for LDA. Gibbs Sampler for LDA - Agustinus Kristiadi's Blog. https://wiseodd.github.io/techblog/2017/09/07/lda-gibbs/. Accessed 19 November 2019.

[14] Kumar, P. (2017, October 21). An Introduction to N-grams: What Are They and Why Do We Need Them? XRDS. https://blog.xrds.acm.org/2017/10/introduction-n-grams-need/. Accessed 19 November 2019.

[15] Liu, S. (2019, January 11). Dirichlet distribution. Medium. Towards Data Science. https://towardsdatascience.com/dirichlet-distribution-a82ab942a879. Accessed 19 November 2019.

[16] Multinomial distribution. (2019, October 31). Wikipedia. Wikimedia Foundation. https://en.wikipedia.org/wiki/Multinomial_distributionikipedia.org/wiki/Multinomial_distribution. Accessed 19 November 2019.

[17] Murphy, K. P. (2012). Machine learning: a probabilistic perspective. Cambridge, MA: MIT Press.

[18] Pagels, M. (2019, July 4). Introducing one of the best hacks in machine learning: the hashing trick. Medium. Hands-On Advisors. https://medium.com/value-stream-design/introducing-one-of-the-best-hacks-in-machine-learning-the-hashing-trick-bf6a9c8af18f. Accessed 19 November 2019.

[19] Pennsylvania State University. Example of MCMC with full conditional calculations. http://personal.psu.edu/drh20/515/hw/MCMCexample.pdf. Accessed 19 November 2019.

[20] Rathi, A. (2019, July 26). Bayesian Statistics for Data Science. Medium. Towards Data Science. https://towardsdatascience.com/bayesian-statistics-for-data-science-45397ec79c94. Accessed 19 November 2019.

[21] Silge, J., & Robinson, D. (2017). Text mining with R: a tidy approach. Sebastopol, CA: OReilly Media.

[22] Tang, F. (2019, July 14). Beginner's Guide to LDA Topic Modelling with R. Medium. Towards Data Science. https://towardsdatascience.com/beginners-guide-to-lda-topic-modelling-with-r-e57a5a8e7a25. Accessed 19 November 2019.

[23] Twitter. (2017). Twitter Sentiment Analysis, Version 1. Retrieved September.

[24] Wickham, H., & Grolemund, G. (2017). R for data science: import, tidy, transform, visualize, and model data. Sebastopol, CA: OReilly Media.

[25] Ye, X. Study Notes on the Latent Dirichlet Allocation. Johns Hopkins University. http://www.cis.jhu.edu/ xye/papers_and_ppts/ppts/LDA_study_notes_Xugang.pdf. Accessed 19 November 2019.