CHAPTER 5

Bag of Words

In the previous chapter, we covered some of the key biases to watch out for when curating a new collection of documents. In order to apply the tools of text analysis we need to give those documents a numerical representation. In this chapter we introduce the "bag of words" model. This representation provides the foundation for the multinomial language model and the vector space model, which we cover in Chapters 6 and 7 respectively.

As we introduced in Chapter 3, we will use the State of the Union corpus to introduce the bag of words model to show how the expression of the policy priorities of presidents in the US has changed over time. We use this corpus in part to show how to represent texts as data, but we also hope to convey how even simple quantitative analyses of text used in conjunction with deep knowledge of social science can be powerful for social science research.

5.1 The Bag of Words Model

The most common text representation is called the *bag of words* model. The core idea is simple: we will represent each document by counting how many times each word appears in it.

For example, we might be interested in how Presidential focus on the manufacturing sector has changed over time. We take two sentences including the word `manufacturing` from the State of the Union corpus. For now, let's treat each of these short sentences as documents: and represent them in a matrix.

	undoubtedly	power	congress	seriously	affect	agricultural	manufacturing	interests	france	passage	laws	relating	trade	united	states	make	sure	foreign	company	advantage	american	comes	accessing	financing	new	markets	like	russia
Doc 1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Doc 2	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

Example 5.1. Two
SOTU corpus.

Doc 1: It is
seriously t
interests
her trade

Doc 2: And th
company has
when it cor
like Russia

This is a *docum*
and each column
away a lot of infor
a parsimonious re

While the core
decisions that hav
chapter we lay out
introduce some m

1. Choose the
2. Tokenize
3. Reduce Con
   - Lowercase
   - Remove P
   - Remove S
   - Create Eq
   - Filter by F
4. Create the D

5.2 Choose th

After digitizing
cases, the unit of
quantities as topic
posts to journal ar
or larger than the
into headlines and
one day. For our e
short and are there

There are three
of splitting, the m

Documents—e
in the form we wo
the Library of Con

Example 5.1. Two example sentences that include the word `manufacturing` from the SOTU corpus.

> Doc 1: `It is undoubtedly in the power of Congress seriously to affect the agricultural and manufacturing interests of France by the passage of laws relating to her trade with the United States.` (President Jackson, 1831)
>
> Doc 2: `And this Congress should make sure that no foreign company has an advantage over American manufacturing when it comes to accessing financing or new markets like Russia.` (President Obama, 2012)

This is a *document-feature matrix* where each row represents a different document and each column defines a feature that we use to represent the document. This throws away a lot of information—most notably word order—but provides the researcher with a parsimonious representation of the document that is useful for many purposes.

While the core idea of a bag of words is quite simple, there are a multitude of small decisions that have to be made to create a representation like the one above. In this chapter we lay out a default recipe for the bag of words model and in later chapters we introduce some more of the complexities. In total the full recipe is:

1. Choose the Unit of Analysis
2. Tokenize
3. Reduce Complexity
   - Lowercase
   - Remove Punctuation
   - Remove Stop Words
   - Create Equivalence Classes (Lemmatize/Stem)
   - Filter by Frequency
4. Create the Document-Feature Matrix

5.2 Choose the Unit of Analysis

After digitizing the text, the analyst must decide on the *unit* of analysis. In many cases, the unit of analysis is the document—some researchers want to measure such quantities as topics and sentiments in anything from newspaper articles to social media posts to journal articles. However, in some cases the unit of analysis might be smaller or larger than the document—the analyst may want to break apart a newspaper article into headlines and paragraphs, or combine all the tweets written by a single author in one day. For our example, we will use sentences as the unit of analysis because they are short and are therefore more useful for expository purposes.

There are three things to consider when choosing the unit of analysis: the logistics of splitting, the model, and the question.

Documents—especially documents that are difficult to access—don't always come in the form we would ideally like. For example, the *Chronicling America* project from the Library of Congress collects and digitizes historical newspapers from 1789 to 1963.

These documents are scanned from newsprint and then digitized using optical character recognition (OCR) technology. Because newspapers tend to print in columns and news articles often jump across pages, it is difficult to actually access the text that goes with a particular news article even if it is the relevant unit of analysis. Instead, as a compromise to the logistics of splitting up the articles, we might analyze the newspaper page (Young, 2012). Logistical issues also arise when we want to split a text into sections but those sections are not uniformly labeled or otherwise easy to detect.

The second consideration is how the unit of analysis affects the model. We will discuss this issue in more depth in the coming chapters, but in some cases splitting up a single text into many documents can substantially increase or decrease the computational cost of fitting a model. It also may make the model more or less statistically efficient. In the digital humanities, researchers often analyze long texts like novels and split them up into chunks of a few hundred words to improve model fits (Jockers, 2013).

The final—and most important—consideration is the question itself. Just as researchers with a question about when countries go to war should use countries or country dyads rather than provinces as the unit of analysis, researchers interested in how the plot of a novel varies over time should study a novel instead of individual chapters or sentences. This can create a conflict between what is computationally expedient and what best addresses the question of interest. These trade-offs don't always have obvious answers. Sometimes the model can be fit on small chunks of text and the results aggregated to estimate properties of the question-relevant unit, but depending on the model this might not be straightforward. The decisions need to be made on a case-by-case basis.

For the rest of the book, for ease of understanding, we will typically refer to the unit of analysis—whether novels, paragraphs, or tweets —as the "document." We want to be clear that this does not mean we always expect the researcher to be using a full document as the unit of analysis.

5.3 Tokenize

Having chosen the unit of analysis, we are ready to move to the second step, tokenization. While choosing the unit of analysis is about breaking up the corpus into discrete documents, tokenizing is about breaking up a document into discrete words. In our initial example, we split our two short sentences (henceforth "documents") into their constituent words. The bag of words model retains only the number of times each word appears in each document. Each individual word in the document is a *token* and the process of splitting a document into its constituent words is called *tokenization*. More generally, *tokens* are the individual units we split our document into before counting them up. Each token is of a particular *type* (in this context the name of the column). We will often refer to the set of *types* as the vocabulary.

In English, distinct words are separated using white space. This means from a code perspective, tokenization is essentially as simple as splitting the text up by white space. This approach will work for most languages. However, in some languages, such as Chinese, Japanese, and Lao, words are not separated by spaces; instead, words are inferred by the reader from the context within the sentence. For these languages we need a model, called a word segmentation model, to split the characters into their constituent words.

It is worth considering why we choose to divide documents into words at all. On one extreme we could split documents into individual letters. This would have the advantage

of there being only 26 possible types, one for each letter from A to Z (in English at least). This probably wouldn't be a great representation of the document because the letter K has no meaning on its own. On the other extreme we could split documents into individual sentences. Sentences generally contain a complete idea, but any given sentence is probably unique even in a corpus of millions of documents—that is, there are far too many types. Words are a compromise because they individually contain meaning but there are a limited number of types.

Yet sometimes even the single word can be too narrow because some concepts bridge words. A common example is `white house`, where some important meaning is lost when the phrase is tokenized into `white` and `house`. We can recover some of this meaning by using *n*-grams, an ordered set of *n* words, as our features. We call single words *unigrams*, ordered pairs *bigrams*, and ordered triples *trigrams*. Using higher order *n*-grams substantially increases the number of unique types, but can aid our text analysis by retaining more information.

Researchers can include n-grams in two ways. First, they could include all consecutive sets of two words that appear in the corpus, in addition to or instead of unigrams. This will result in a very large number of types and thus many columns in our final document-feature matrix. We will discuss ways to reduce the number of unique types in Section 5.4.5 on filtering.

Alternatively, researchers might find it useful to retain a list of particular bigrams and trigrams that they anticipate will be useful to their analysis. For example, Rule, Cointet, and Bearman (2015) use some multiword phrases such as `national security` and `local government` in their analysis of SOTU speeches. We will visually denote this by placing an underscore between two words that are being treated as an *n*-gram. When the list of *n*-grams to be extracted is small, this solves the problem of the large vocabulary, but it requires an *n*-gram list.

A common way to produce lists of *n*-grams (and also the approach taken by Rule, Cointet, and Bearman (2015)) is based on statistical tests. There are many variants but they share a common logic. Imagine that we are trying to create a list of bigrams that have a meaning separate from their constituent parts. We can use the probability of seeing this pair of words relative to the probability of seeing them each independently. For example, if the chance of seeing `white house` is high enough relative to seeing `white` and `house` separately, we add it to our list of bigrams. There are other strategies for identifying *n*-grams that are based on grammatical patterns in

Example 5.2. Example of tokenization for a sentence from the SOTU corpus. `United States` is converted to `United_States` to make it a bigram.

```
It is undoubtedly in the power of Congress seriously
to affect the agricultural and manufacturing interests
of France by the passage of laws relating to her trade
with the United States.

It is undoubtedly in the power of Congress seriously
to affect the agricultural and manufacturing interests
of France by the passage of laws relating to her trade
with the United_States.
```

language. We defer a discussion of these until Chapter 9 on structured representations of language.

In Example 5.2, we show how the first document from Example 5.1 could be tokenized. We have included the bigram `United States` as an *n*-gram that we want to treat as a single token.

5.4 Reduce Complexity

Once the words in the document have been tokenized, we could immediately jump to the fourth step of the recipe and count them up and construct our document-feature matrix. However, for many applications this will lead to a vocabulary that is far too large to be productive. Thus, in bag of words models there is usually a stage where we drop some word types in order to reduce complexity. Reducing complexity will help us with computation and make downstream analysis parsimonious. In this section we identify five common steps and identify the many decisions that go into each one. At each stage we update our running example.

5.4.1 LOWERCASE

Often the first step after tokenizing identifies the individual words is to replace all capital letters with lowercase letters. The idea is that the word `In` is no different than `in` for most applications and research questions. A variation of this strategy is to only lowercase words when they begin a sentence.

Original: `It is undoubtedly in the power of Congress seriously to affect the agricultural and manufacturing interests of France by the passage of laws relating to her trade with the United States.`

Before Lowercase: `It is undoubtedly in the power of Congress seriously to affect the agricultural and manufacturing interests of France by the passage of laws relating to her trade with the United_States.`

After Lowercase: `it is undoubtedly in the power of congress seriously to affect the agricultural and manufacturing interests of france by the passage of laws relating to her trade with the united_states.`

5.4.2 REMOVE PUNCTUATION

The second step is to remove all punctuation. Again for many research questions, researchers are often not interested in how many commas, apostrophes, or periods appear within the text, and thus choose to delete them. The researcher may also want to get rid of formatting cues that often exist within data from other sources. For example, section headings, page numbers, and html tags, may not be important to the purpose of the research, and, if so, should be discarded before the analysis. On the other hand, certain forms of punctuation, for example the # symbol on Twitter,

may be essential to retain the meaning of the original work. For certain tasks, such as sentiment analysis, this step must be done carefully as common punctuation-based emojis, such as `:)`, and exclamation marks can be strong predictors of sentiment. Note that we are careful here not to remove the underscore as we are using it to indicate the bigram.

Original: `It is undoubtedly in the power of Congress seriously to affect the agricultural and manufacturing interests of France by the passage of laws relating to her trade with the United States.`

Before Punctuation Removal: `it is undoubtedly in the power of congress seriously to affect the agricultural and manufacturing interests of france by the passage of laws relating to her trade with the united_states.`

After Punctuation Removal: `it is undoubtedly in the power of congress seriously to affect the agricultural and manufacturing interests of france by the passage of laws relating to her trade with the united_states`

5.4.3 REMOVE STOP WORDS

After lowercasing and removing unwanted punctuation, the only content left in the document is the words themselves. The analyst may decide to remove *stop words*—common words used across documents that do not give much information about the task at hand.[1] In English, common words such as `and`, `the`, and `that` may be removed

Original: `It is undoubtedly in the power of Congress seriously to affect the agricultural and manufacturing interests of France by the passage of laws relating to her trade with the United States.`

Before Stop Word Removal: `it is undoubtedly in the power of congress seriously to affect the agricultural and manufacturing interests of france by the passage of laws relating to her trade with the united_states`

After Stop Word Removal: `undoubtedly power congress seriously affect agricultural manufacturing interests france passage laws relating trade united_states`

[1]The idea of omitting a specific list of common words appears to have originated in the early information retrieval work of Harold Luhn while proposing keywords in context (Luhn, 1960). Barbara Flood recounts that in 1961 she was compiling biological abstracts and they had decided to try the new keywords in context approach to indexing. She was given reams of printouts to look through manually and decided to delete highly common words in the index such as `of` and `in`. Because these were going to be literally stopped from printing, she called them the "stop list" (Flood, 1999).

from the documents to reduce the size and complexity of the feature set. Removing stop words often removes an extremely high fraction of the corpus's *tokens* while removing very few of the *types*, which—if those words carry little or no meaning—can result in huge computational savings. This reflects the empirical fact that in most languages a few words are extremely common. The idea is that even though these tokens account for a large fraction of the words, they account for only a small fraction of the meaning. Lists of stop words in many languages are available in software packages that provide text analysis tools.

5.4.4 CREATE EQUIVALENCE CLASSES (LEMMATIZE/STEM)

Shifting everything to lower case, removing punctuation, and dropping stop words will substantially reduce the number of types in the vocabulary. However, there can still be cases where many words carry the same information because they share a common root. For example `family`, `families`, and `family's` (with the apostrophe possibly dropped) are all distinct types in the vocabulary but for some tasks it may be effective to map them all to a common form. A *lemma* is the canonical form (such as one might find in a dictionary) of a set of words that are related by inflection (i.e., modifications due to case, number, tense, etc.). Lemmatization is the process of mapping words to their lemma. This is an equivalence assertion—a statement that for our purposes we will treat all variants of the word the same.

Sometimes identifying the lemma is relatively simple, as in the case of variants of `family`, but it can also be quite complex. For example, in English, many common words (e.g., `see`) have an irregular past tense (e.g., `saw`). Even exactly the same token can have two different lemmas depending on the part of speech. For example, a piece of writing would have the lemma `writing` while the act of writing would have the lemma `write`. Lemmatization can be cumbersome because it often requires a dictionary lookup (to map `saw` to `see`) and part of speech tagging (to distinguish the two lemmas of `writing`). A popular approximation to lemmatizing that also maps related forms together is *stemming*. In stemming, we simply discard the end of a word using a few simple rules. In the case of `family`, all of our variants above would be mapped to `famili`. While stemming is often quite effective and substantially faster than lemmatization, it does fail to deal with words that have more complex forms

```
It is undoubtedly in the power of Congress seriously
to affect the agricultural and manufacturing interests
of France by the passage of laws relating to her trade
with the United States.

Before Stemming: undoubtedly power congress seriously
affect agricultural manufacturing interests france
passage laws relating trade united_states

After Stemming: undoubt power congress serious affect
agricultur manufactur interest franc passag law relat
trade united_states
```

(such as the `see` and `writing` examples). Stemming can also sometimes reduce two words to a common stem that may have quite distinct meanings such as `secure` and `securities`.

There are many stemming algorithms available in a wide range of software; among the most common is the *Porter* stemming algorithm (Porter, 1980). Stemming is still substantially more popular than lemmatization, likely due to a combination of speed, accessibility across languages, and momentum. Depending on the language you are working in and the computational resources available, lemmatization may be more or less preferred to stemming. In either case, the goal is to simplify the analysis by bringing down the number of types by treating the variants of a word as equivalent in our document-feature matrix.

5.4.5 FILTER BY FREQUENCY

The final step of reducing complexity before creating the document-feature matrix is to remove words that are very rare (or sometimes, very common). If a word appears only once in the corpus, we are unlikely to use it effectively and, depending on the model, removing all such words from the corpus might yield substantial computational savings. Due to the properties of language there are often many such rare words in a corpus.

In other cases we may want to remove words that have incredibly high frequency. A word that is used in every document is unlikely to be useful in helping us discriminate between documents. We like to think of these as application-specific stop words and they often appear when the corpus was explicitly selected on presence of a keyword or when the corpus is on a particular topic with a common vocabulary.

In total, the recipe we have given is tokenize, lowercase, remove punctuation, create equivalence classes, and filter by frequency. It is worth noting that the order in which these steps are taken matters as well. For example, if stop words are removed before tokenization, the analyst will end up with different bigrams and trigrams than if they are removed before. The analyst must use her best judgement to decide which preprocessing steps to take and in what order. When the preprocessed data are used in analyses downstream, validation will be needed to ensure that the preprocessing retained the information relevant to the researcher's question of interest. We will talk more extensively about validation in later parts of the book.

5.5 Construct Document-Feature Matrix

Now that we have tokenized the corpus and reduced its complexity somewhat, it is time to turn the text into numeric data. We will use N to delineate the number of unique text units within your dataset, and i to index each of these units. After preprocessing these units, you can identify the number of types within your dataset, which is the size of your vocabulary and which we will delineate with J and index by j. The result of this is what we call a document-feature matrix, which we will call W, an $N \times J$ matrix. Each element W_{ij} of this matrix is the number of times that each type appears in the document. In general, since most documents only contain a few of the many unique types within a corpus, the document-feature matrix is relatively *sparse*—that is, it contains mostly zeros.

We return to our two example sentences from the SOTU corpus, below:

SOTU example corpus

Doc 1: `It is undoubtedly in the power of Congress seriously to affect the agricultural and manufacturing interests of France by the passage of laws relating to her trade with the United States.` (President Jackson, 1831)

Doc 2: `And this Congress should make sure that no foreign company has an advantage over American manufacturing when it comes to accessing financing or new markets like Russia.` (President Obama, 2012)

We could then put them through the same preprocessing routine as we did above. This would create two documents, each having been converted to lowercase, had punctuation removed, had stop words removed, and been stemmed. After doing this, we would document the number of unique types within the corpus, the *vocabulary*. Overall, after being preprocessed, these three documents have 28 unique words. W would therefore consist of two rows (one for each document) and 28 columns (one for each unique word). The entries of those rows would count the number of times each word appeared in each document; see below.

Once we have W, there are many operations we can perform on it that can help summarize the text. To take two simple examples to start, if we sum each of the columns of W, we will have a summary of how many of each unique type appear within the corpus.

Or, we could sum each of the rows of the document-feature matrix, which would tell us how many tokens each document has (after preprocessing). The maximum of this sum would be the length of the longest document, and the minimum of this sum would

Table 5.1. Document-feature matrix W from two sentences containing the word `manufacturing` from the SOU corpus.

	undoubt	power	congress	serious	affect	agricultur	manufactur	interest	franc	passag	law	relat	trade	unit	state	make	sure	foreign	compani	advantag	american	come	access	financ	new	market	like	russia
Doc 1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Doc 2	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 5.2. Result of summing the columns of W, the document-feature matrix.

undoubt	power	congress	serious	affect	agricultur	manufactur	interest	franc	passag	law	relat	trade	unit	state	make	sure	foreign	compani	advantag	american	come	access	financ	new	market	like	russia
1	1	2	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 5.3. Result of summing the rows of W, the document-feature matrix.

Doc 1	15
Doc 2	15

be the length of the shortest document. While these are basic examples, much of the rest of this book describes algorithms that perform operations on representations like this document-feature matrix.

5.6 Rethinking the Defaults

The default procedure that we outline above is simply that, a default procedure. It doesn't work for all cases and is primarily designed for settings where our task is to cover the general subject matter of the documents, as in the examples from the SOTU corpus. There are two types of settings where you may want to abandon one or more of these procedures. The first is where the procedures are inappropriate for the type of problem and the second is where the data is sufficiently large that the procedures are harmful.

5.6.1 AUTHORSHIP OF THE FEDERALIST PAPERS

To see how the default procedure we describe can be inappropriate for particular problems, we return to Mosteller and Wallace (1963)'s analysis of the Federalist Papers that we described in Chapter 3. Mosteller and Wallace (1963) tackle the problem of unknown authorship of a set of the Federalist Papers, documents written in support of the American Constitution by a combination of Alexander Hamilton, John Jay, and James Madison in the late 1700s. While the authors of many of the papers are known, the authorship of 12 particular papers is disputed.

Mosteller and Wallace (1963) show that the writing style of Jay, Hamilton, and Madison differed significantly through their use of *filler words*—particularly words like `upon`, `by`, and `to`. These are precisely the kinds of words we would filter out as stop words because they don't contain any meaningful content about subject matter. In this context though, that was exactly the objective. In his autobiography, Mosteller recalled that they wanted "words whose use is unrelated to the topic and may be regarded as reflecting minor or perhaps unconscious preferences of the author" (Mosteller, 2010, p. 56).

This kind of conflict between the default recipe and substantive problem is restricted not just to the author attribution problem. Social scientists of all types have shown the importance of text features that the default recipe can treat as content-free (Biber, 1991; Chung and Pennebaker, 2007; Breiger, Wagner-Pacifici, and Mohr, 2018). For example, Monroe, Colaresi, and Quinn (2008) show that gender pronouns like `he` and `she` are highly associated with the political party of the speaker in speech in the US Congressional Record. Since these pronouns are included in nearly every English stop word list, we might miss a valuable insight by just dropping all stop words without thinking about it. Denny and Spirling (2018) demonstrate that variations in many of the steps we describe above can impact the results of downstream analyses and suggest

Figure 5.1. Word clouds of the most common words in the sentences in the SOTU corpus containing the word `manufacturing`. The left panel shows the word cloud with only punctuation removed. The right panel shows the word cloud with the default recipe. In both cases, we remove the word `manufacturing` since it is most prevalent by design.

testing several different combinations to see if the outcomes differ. We will return to this argument later in the book.

5.6.2 THE SCALE ARGUMENT AGAINST PREPROCESSING

Many of the steps in the default recipe are about reducing the size of the vocabulary. There are essentially two reasons for this: aesthetics and efficiency.

When we make word clouds or visualize model results, not removing stop words can make those visualizations less informative. Figure 5.1 shows two word clouds of the SOTU corpus sentences that contain the word `manufacturing`. The left panel does not do any removal besides punctuation, and the right panel follows the default recipe. We see immediately that the right panel is much more informative about the subjects of the sentences that contain the word `manufacturing`, while the left panel highlights common and uninformative words such as `of` and `the`. While the aesthetic consideration is an an important practical reality, it can be addressed by changing our visualizations rather than changing our inputs. For example, we could always include stop words in our document-feature matrix but explicitly omit them from the visualization. When we discuss separating words in the Discovery part of the book, we will return to this issue of how to find representative words that are less susceptible to the clutter of uninformative words.

The second reason to reduce the size of the vocabulary is efficiency of the subsequent data analysis. Stemming and lemmatizing make an assertion that two related words like `car` and `cars` can be treated exactly the same. If they have close enough to a common meaning, baking in that knowledge will be helpful because it means fewer types in the vocabulary to deal with. However, the salience of this advice arises from a pragmatically understandable focus on corpora with relatively few documents. Yet, as social scientists move from analyzing collections of thousands documents to routinely analyzing collections of millions of documents, it may not be as important to assert that any subsequent analysis treat `car` and `cars` the same because our models have enough

information to figure out these relations on their own. Indeed, we could even be wrong about the two words conveying similar information. If we are predicting the wealth of the author, mentioning `cars` may mean something different on average than `car`.

A recent line of work by Alexandra Schofield and David Mimno has documented that for topic models—a popular class of models that we will discuss in Discovery where these procedures are almost always applied—the application of default stemming and stop word removal can actually be harmful for model performance when the corpus is large (Schofield and Mimno, 2016; Schofield, Magnusson, and Mimno, 2017). The more general lesson here is that folk wisdom and heuristic advice of the sort represented in default recipes is always a product of its time. It is a sign of text analysis maturing as a field that the community is beginning to reexamine those defaults.

5.7 Conclusion

In this chapter we walked step by step through the basic recipe of the bag of words representation of text. While any default recipe will have some limitations—and we explored some of those in the previous section—the bag of words model is shockingly powerful for something that throws away so much information! In the next two chapters, we present two different frameworks for building methods on top of the bag of words representation, the multinomial language model and the vector space model.