

# Conceptual Foundations of Text Mining and Preprocessing Steps

## CONTENTS

Preamble .....	43
Introduction .....	43
Syntax versus Semantics .....	44
The Generalized Vector-Space Model .....	45
Preprocessing Text .....	46
Creating Vectors from Processed Text .....	50
Summary .....	51
Postscript .....	51
Reference .....	51

## PREAMBLE

After you determine what you want to do with text mining, you must compose the set of steps that you can follow to perform it. In the next chapter, you will learn about one of the most important innovations in text processing: the generalized vector-space model. Documents are represented by a string of values (a *vector*), in which each element in the string represents a unique word or word group contained among the list documents. Each element of the vector contains either a 1 or a 0 (for the presence or absence of the word in the document), or a count of the number of times the word is present in a document. When you have a data structure that can be analyzed numerically, you are ready to hit the data mining “trail.”

## INTRODUCTION

Before starting a text mining project, it is first necessary to understand the conceptual foundations of text mining and then to understand how to get started leveraging the power of your data to drive decision making in your organization. This chapter highlights many of the theoretical foundations of text mining algorithms and describes the basic preprocessing steps used to prepare data for text mining algorithms.

The majority of text data encountered on a daily basis is *unstructured*, or free, text. This is the most familiar type of text, appearing in everyday sources such as books, newspapers, emails, and web pages. By unstructured, we mean that this text exists “in the wild” and has not been processed into a structured format, such as a spreadsheet or relational database. Often, text may occur as a field in a spreadsheet or database alongside more traditional structured data. This type of text is called *semistructured*. Unstructured and semistructured text composes the majority of the world’s data; recent estimates place text data at 75 to 80 percent of the data in the world. This is easy to believe considering the amount of information stored as text on the Internet alone.

Given the sheer volume of text available, it is necessary to turn to automated means to assist humans in understanding and exploiting available text documents. This chapter provides a brief introduction to the statistical and linguistic foundations of text mining and walks through the process of preparing unstructured and semistructured text for use with text mining algorithms and software.

## SYNTAX VERSUS SEMANTICS

The purpose of text mining algorithms is to provide some understanding of how the text is processed without having a human read it. However, a computer can only examine directly the individual characters in each word and how those words are arranged. A computer cannot know what information is being communicated by the text, only the structure or syntax of the text. *Syntax* pertains to the structure of language and how individual words are composed to make well-formed sentences and paragraphs. This is an ordered process. Specific grammar rules and language conventions govern how language is used, leading to statistical patterns appearing frequently in large amounts of text. This structure is relatively easy for a computer to process. On its own, however, syntax is insufficient for fully understanding meaning.

*Semantics* refers to the meaning of the individual words within the surrounding context. Common idioms are useful to illustrate the differences between syntax and semantics. Idioms are words or phrases with a figurative meaning that is different from the literal meaning of the words. For example, the sentence “Mary had butterflies in her stomach before the show” is syntactically correct and has two potential semantic meanings: a literal interpretation where Mary’s preshow ritual includes eating butterflies and an idiomatic interpretation that Mary was feeling nervous before the show. Clearly, the complete semantic meaning of the text can be difficult to determine automatically without extensive understanding of the language being used.

Fortunately, syntax alone can be used to extract practical value from the text without a full semantic understanding, due to the close tie between syntax and semantics. Many text mining tasks, such as *document classification* and *information retrieval*, are concerned with ranking or finding specific types of documents in a large document database. The main assumption behind these algorithms is that syntactic similarity (similar words) implies semantic similarity (similar meaning). Though relying on syntactic information alone, these approaches work because documents that share many keywords are often on the same topic. In other instances, the goal is really about semantics. For example, *concept extraction* is about automatically identifying words and phrases that have the same meaning. Again, the text mining approaches must rely on the syntax to infer a semantic relationship. The generalized vector-space model, described next, enables this.

## THE GENERALIZED VECTOR-SPACE MODEL

The most popular structured representation of text is the vector-space model, which represents text as a vector where the elements of the vector indicate the occurrence of words within the text. This results in an extremely high-dimensional space; typically, every distinct string of characters occurring in the collection of text documents has a dimension. This includes dimensions for common English words and other strings such as email addresses and URLs. For a collection of text documents of reasonable size, the vectors can easily contain hundreds of thousands of elements. For those readers who are familiar with data mining or machine learning, the vector-space model can be viewed as a traditional feature vector where words and strings substitute for more traditional numerical features. Therefore, it is not surprising that many text mining solutions consist of applying data mining or machine learning algorithms to text stored in a vector-space representation, provided these algorithms can be adapted or extended to deal efficiently with the large dimensional space encountered in text situations.

The vector-space model makes an implicit assumption (called the *bag-of-words* assumption) that the order of the words in the document does not matter. This may seem like a big assumption, since text must be read in a specific order to be understood. For many text mining tasks, such as *document classification* or *clustering*, however, this assumption is usually not a problem. The collection of words appearing in the document (in any order) is usually sufficient to differentiate between semantic concepts. The main strength of text mining algorithms is their ability to use *all* of the words in the document—primary keywords and the remaining general text. Often, keywords alone do not differentiate a document, but instead the usage patterns of the secondary words provide the differentiating characteristics.

Though the bag-of-words assumption works well for many tasks, it is not a universal solution. For some tasks, such as *information extraction* and *natural language processing*, the order of words is critical for solving the task successfully. Prominent features in both entity extraction (see Chapter 13) and natural language processing include both preceding and following words and the decision (e.g., the part of speech) for those words. Specialized algorithms and models for handling sequences such as finite state machines or *conditional random fields* are used in these cases.

Another challenge for using the vector-space model is the presence of *homographs*. These are words that are spelled the same but have different meanings. One example is the word *saw*, which can be a noun describing a tool for cutting wood (e.g., “I used a saw to cut down the tree”) or the past tense of the verb *to see* (e.g., “I saw the tree fall”). Fortunately, homographs do not typically have a large effect on the results of text mining algorithms for three reasons:

1. Using multiple senses of a single homograph in a document is rare.
2. If used, homographs are typically not the most meaningful feature for the task.
3. Other words appearing in the document are usually sufficient to distinguish between different use cases (for example, the word *cut*).

Though the situation is rare, it is possible for homographs to result in text with completely different meanings to be grouped together in vector space. One extreme example is the following pair of sentences:

“She can refuse to overlook our row,” he moped, “unless I entrance her with the right present:  
a hit.”

Her moped is presently right at the entrance to the building; she had hit a row of refuse cans!

“She can refuse **to** overlook our row,” he moped, “unless I entrance her **with** **the** right present: **a** hit.”

Her moped **is** presently right **at** **the** entrance **to** **the** building; she had hit **a** row **of** refuse cans!

**FIGURE 3.1**

Example sentences with the nine homographs underlined, the stopwords in **bold**, and two stemmed words in *italics*. If you ignore the stopwords, there are 11 shared words, and the sets of distinctive words are only {he, I, our, unless} versus {had}.

These two sentences are nearly the same according to the vector-space model. That is, they are nearly identically located in high-dimensional word space. The two sentences are even somewhat more similar than they at first appear, as shown in Figure 3.1. First, most text preprocessing uses a normalization process called *stemming* to remove pluralization and other suffixes, which would make “can” and “cans” identical in the sentences, as well as “present” and “presently.” The two sentences thus share homographs. They also share a few other words (e.g., “she,” “her”), increasing their similarity in a normal way, and some of their distinct words—which would otherwise increase diversity—are *stopwords* (such as “at” and “with”), which are removed during processing and not used in the vector space model. So they are distinguished only by the words {he, I, our, unless} vs. {had}, despite depicting very different scenes. The preprocessing steps just mentioned are described in more detail next.

## PREPROCESSING TEXT

One of the challenges of text mining is converting unstructured and semistructured text into the structured vector-space model. This must be done prior to doing any advanced text mining or analytics. The possible steps of text preprocessing are the same for all text mining tasks, though which processing steps are chosen depends on the task. The basic steps are as follows:

1. Choose the scope of the text to be processed (documents, paragraphs, etc.).
2. Tokenize: Break text into discrete words called *tokens*.
3. Remove stopwords (“stopping”): Remove common words such as *the*.
4. Stem: Remove prefixes and suffixes to normalize words—for example, *run*, *running*, and *runs* would all be stemmed to *run*.
5. Normalize spelling: Unify misspellings and other spelling variations into a single token.
6. Detect sentence boundaries: Mark the ends of sentences.
7. Normalize case: Convert the text to either all lower or all upper case.

### Choosing the Scope of Documents

For many text mining tasks, the scope of the text is easy to determine. For example, emails or call log records naturally translate into a single vector for each message. However, for longer documents one needs to decide whether to use the entire document or to break the document up into sections, paragraphs, or sentences. Choosing the proper scope depends on the goals of the text mining task: for *classification* or *clustering* tasks, often the entire document is the proper scope; for *sentiment analysis*,

*document summarization*, or *information retrieval*, smaller units of text such as paragraphs or sections might be more appropriate.

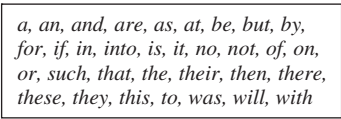
## Tokenization

The next preprocessing step is breaking up the units of text into individual words or tokens. This process can take many forms, depending on the language being analyzed. For English, a straightforward and effective tokenization strategy is to use white space and punctuation as token delimiters. This strategy is simple to implement, but there are some instances where it may not match the desired behavior. In the case of acronyms and abbreviations, the combination of using an unordered vector space and separating tokens on punctuation would put different components of the acronym into different tokens. For example U.N. (abbreviation for the United Nations) would be tokenized into separate tokens U and N losing the meaning of the acronym. There are two approaches to address this problem: smart tokenization, which attempts to detect acronyms and abbreviations and avoid tokenizing those words, and adjusting the downstream processing to handle acronyms (for example) in separate tokens prior to converting to a vector space.

## Stopping

For many text mining tasks, it is useful to remove words such as *the* that appear in nearly every document in order to save storage space and speed up processing. These common words are called “stopwords,” and the process of removing these words is called “stopping.” An example stopword list from a popular open-source text mining library is shown in [Figure 3.2](#). Stopping is a commonly included feature in nearly every text mining software package. The removal of stopwords is possible without loss of information because for the large majority of text mining tasks and algorithms, these words have little impact on the final results of the algorithm. Text mining tasks involving phrases, such as *information retrieval*, are the exception because phrases lose their meaning if some of the words are removed.

A special case of stopwords removal is the detection of header and “boilerplate” information. For example, many corporate emails also include a lengthy legal disclaimer at the bottom of the message. For short messages, a long disclaimer can overwhelm the actual text when performing any sort of text mining. In the same way, email header information such as “Subject,” “To,” and “From,” if not removed, can lead to skewed results. Removal of this type of text depends on the situation where it appears.



*a, an, and, are, as, at, be, but, by,  
for, if, in, into, is, it, no, not, of, on,  
or, such, that, the, their, then, there,  
these, they, this, to, was, will, with*

**FIGURE 3.2**

English stopwords used by the open-source Lucene search index project.

## Stemming

Stemming is the process of normalizing related word tokens into a single form. Typically the stemming process includes the identification and removal of prefixes, suffixes, and inappropriate pluralizations.

For example, a typical stemming algorithm would normalize *walking*, *walks*, *walked*, *walker*, and so on into *walk*. For many text mining tasks, including classification, clustering, or search indexing, stemming leads to accuracy improvements by shrinking the number of dimensions used by the algorithms and grouping words by concept. This decrease in dimensionality improves the operation of the algorithms.

One of the most popular stemming algorithms, the Porter stemmer, used a series of heuristic replacement rules to transform word tokens into their stemmed form (Porter, 1980).<sup>1</sup> These rules have been refined in later versions into what is now known as the Snowball Stemmer. The Snowball Stemmer is available in nearly 20 different languages and is a major component of the popular Lucene search engine index released by the Apache Software Foundation.<sup>2</sup> As with most text applications, the general case works well but requires some special cases that the Porter Stemmer calls “dictionary lookups.” For example, words ending in *y* are often stemmed as *ability* → *abilit*, but *sky* loses its meaning when the *y* is removed and requires a special case. Similarly, the suffix *-ly* (as in *quickly*) is usually removed, but for words such as *reply*, removing *-ly* is not appropriate.

Lemmatization is a more advanced form of stemming that attempts to group words based on their core concept or *lemma*. Lemmatization uses both the context surrounding the word and additional grammatical information such as part of speech to determine the lemma. Consequently, lemmatization requires more information to perform accurately. For words such as *walk*, stemming and lemmatization produce the same results. However, for words like *meeting*, which could serve as either a noun or a verb, stemming produces the same root *meet*, but lemmatization produces *meet* for the verb and maintains *meeting* in the noun case.

## Spelling Normalization

Misspelled words can lead to an unnecessary expansion in the size of the vector space needed to represent a document. In a recent project involving hand-entered text, we found over 50 different spellings of the phrase “learning disability” (Figure 3.3). There are a number of different approaches for correcting spelling automatically. First, dictionary-based approaches can be used to fix common spelling variations such as differences between American and British English (e.g., color and colour). Second, fuzzy matching algorithms such as soundex, metaphone, or string-edit distance can be used to cluster together words with similar spellings. String hashing functions such as soundex or metaphone are found in popular database software and text mining software. Finally, if none of those approaches is sufficient, word clustering and concept expansion techniques (see Chapters 8 and 13) can be used to cluster misspellings together based on usage. These approaches can also be used in combination; for example, if two tokens are used in similar contexts (word clustering) and have a small string-edit distance or similar soundex, then they are likely to be the same word despite variations in spelling. Spelling normalization may not be required if the text is mostly clean and misspellings are rare. For messy text, such as text gathered from the Internet, spelling normalization is invaluable.

## Sentence Boundary Detection

Sentence boundary detection is the process of breaking down entire documents into individual grammatical sentences. For English text, it is almost as easy as finding every occurrence of punctuation

<sup>1</sup> Also available at <http://tartarus.org/~martin/PorterStemmer/def.txt>

<sup>2</sup> <http://lucene.apache.org>

<i>learning disablitiy, learning deisability,</i>
<i>learning disability, learing disabilities,</i>
<i>learning disabiality, learning disability,</i>
<i>learning disabilty, learning disability,</i>
<i>learning disabilty, learning disablety,</i>
<i>learning disability, learnoing disability,</i>
<i>learning disabilties, learning disability,</i>
<i>learning disability, learning disibility,</i>
<i>learnings disability, learningdisability,</i>
<i>larning disabilities, learning disabilities,</i>
<i>learning disabilities, learning disabilities,</i>
<i>learning diasability, learning dasability,</i>
<i>learnning disability, learning disabilities,</i>
<i>lerning disability, learning disabilities,</i>
<i>learneing disability, learnining disability,</i>
<i>learning disaibilities, leraning disability,</i>
<i>learning disaiblity, learnings disability,</i>
<i>learning disabilities, learning disabilty,</i>
<i>learnings disabilities, learning diasbility,</i>
<i>learning disabliites, learning dsiability,</i>
<i>learning disablity, learning disibilty,</i>
<i>learning disibilities, learning disbalty,</i>
<i>learning disability, learning disabilities,</i>
<i>learning disabilities, learningi disability,</i>
<i>lerniung disabilities, learning disabilities,</i>
<i>learning disaability, learning disabilities</i>

**FIGURE 3.3**

Evidence that spelling normalization is needed. This table shows 52 misspellings of the phrase “learning disability” found during a recent text mining project.

like “.”; “?”; or “!” in the text. However, some periods occur as part of abbreviations or acronyms (as noted before on tokenization). Also, quotations will often contain sentence boundary punctuation, but the actual sentence will end later. These conditions suggest a few simple heuristic rules that can correctly identify the majority of sentence boundaries. To achieve near perfect (~99 percent) accuracy, statistical classification techniques are used. More on text classification techniques can be found in Chapter 7.

## Case Normalization

Most English texts (and other Romance languages) are written in mixed case—that is, text contains both upper- and lowercase letters. Capitalization helps human readers differentiate, for example, between nouns and proper nouns and can be useful for automated algorithms as well (see Chapter 9 on named

entity extraction). In many circumstances, however, an uppercase word at the beginning of the sentence should be treated no differently than the same word in lower case appearing elsewhere in a document. Case normalization converts the entire document to either completely lower case or completely upper case characters.

## CREATING VECTORS FROM PROCESSED TEXT

After text preprocessing has been completed, the individual word tokens must be transformed into a vector representation suitable for input into text mining algorithms. This vector representation can take one of three different forms: a binary representation, an integer count, or a float-valued weighted vector. Following is a (very) simple example that highlights the difference among the three approaches. Assume a collection of text with the following three documents:

Document 1: My dog ate my homework.  
 Document 2: My cat ate the sandwich.  
 Document 3: A dolphin ate the homework.

The vector space for these documents contains 15 tokens, 9 of which are distinct. The terms are sorted alphabetically with total counts in parentheses:

a (1), ate (3), cat (1), dolphin (1), dog (1), homework (2), my (3), sandwich (1), the (2).

The binary and integer count vectors are straightforward to compute from a token stream. A binary vector stores a 1 for each term that appears in a document, whereas a count vector stores the count of that word in the document.

Doc 1: 0,1,0,0,1,1,1,0,0 (notice that though “my” appears twice in the sentence, the binary vector still only contains a “1”)  
 Doc 2: 0,1,1,0,0,0,1,1,1  
 Doc 3: 1,1,0,1,0,1,0,0,1

The integer count vectors for the three documents would look as follows:

Doc 1: 0,1,0,0,1,1,2,0,0 (notice that “my” appearing twice in the sentence is reflected)  
 Doc 2: 0,1,1,0,0,0,1,1,1  
 Doc 3: 1,1,0,1,0,1,0,0,1

Storing text as weighted vectors first requires choosing a weighting scheme. The most popular scheme is the TF-IDF weighting approach. TF-IDF stands for *term frequency–inverse document frequency*. The term frequency for a term is the number of times the term appears in a document. In the preceding example, the term frequency in Document 1 for “my” is 2, since it appears twice in the document. Document frequency for a term is the number of documents that contain a given term; it would also be 2 for “my” in the collection of the three preceding documents. Equations for these values are shown in [Figure 3.4](#).

The assumption behind TF-IDF is that words with high term frequency should receive high weight unless they also have high document frequency. The word *the* is one of the most commonly occurring words in the English language. *The* often occurs many times within a single document, but it also occurs in nearly every document. These two competing effects cancel out to give *the* a low weight.



$$tf \bullet idf(t, d) = tf(t, d) \bullet idf(t)$$

$$tf(t, d) = \sum_{i \in d}^{|d|} 1\{d_i = t\}$$

$$idf(t) = \log \left( \frac{|D|}{\sum_{d \in D}^{|\mathcal{D}|} 1\{t \in d\}} \right)$$

**FIGURE 3.4**

Equations for TF-IDF: term frequency (TF)—inverse document frequency (IDF).

## SUMMARY

Once the input text has been processed and converted into text vectors, you are ready to start getting results from applying a text mining algorithm, as described in the remainder of the book. Even though you may seek to discover the semantics or meaning of the text through text mining, algorithms can only directly use the syntax or structure of the document. Fortunately, syntax and semantics are usually closely tied together, and text mining can work well in practice.

## POSTSCRIPT

Similar to the early days of data mining, the term *text mining* can refer to many very different methods that can be applied to many use cases. Before you can start down the text mining trail, it is best to understand how the various use cases are similar and how they are different. It is a waste of time and resources to think that your application will involve specific methods of data preparation when your application really belongs in a different category of use cases. The purpose of the next chapter is to pair the appropriate use cases with the appropriate technologies to accomplish them.

## Reference

Porter, M. F. 1980. An algorithm for suffix stripping. *Program* 14(3): 130–137.