SAMPLE CHAPTER

Taming Text

How to find, organize, and manipulate it

Grant S. Ingersoll
Thomas S. Morton
Andrew L. Farris

FOREWORD BY Liz Liddy

MANNING

Taming Text

by Grant S. Ingersoll
Thomas S. Morton
Andrew L. Farris

Chapter 1

brief contents

1 Getting started taming text

In this chapter

- Understanding why processing text is important
- Learning what makes taming text hard
- Setting the stage for leveraging open source libraries to tame text

If you're reading this book, chances are you're a programmer, or at least in the information technology field. You operate with relative ease when it comes to email, instant messaging, Google, YouTube, Facebook, Twitter, blogs, and most of the other technologies that define our digital age. After you're done congratulating yourself on your technical prowess, take a moment to imagine your users. They often feel imprisoned by the sheer volume of email they receive. They struggle to organize all the data that inundates their lives. And they probably don't know or even care about RSS or JSON, much less search engines, Bayesian classifiers, or neural networks. They want to get answers to their questions without sifting through pages of results. They want email to be organized and prioritized, but spend little time actually doing it themselves. Ultimately, your users want tools that enable

them to focus on their lives and their work, not just their technology. They want to control—or tame—the uncontrolled beast that is *text*. But what does it mean to tame text? We'll talk more about it later in this chapter, but for now taming text involves three primary things:

- The ability to find relevant answers and supporting content given an information need
- The ability to organize (label, extract, summarize) and manipulate text with little-to-no user intervention
- The ability to do both of these things with ever-increasing amounts of input

This leads us to the primary goal of this book: to give you, the programmer, the tools and hands-on advice to build applications that help people better manage the tidal wave of communication that swamps their lives. The secondary goal of *Taming Text* is to show how to do this using existing, freely available, high quality, open source libraries and tools.

Before we get to those broader goals later in the book, let's step back and examine some of the factors involved in text processing and why it's hard, and also look at some use cases as motivation for the chapters to follow. Specifically, this chapter aims to provide some background on why processing text effectively is both important and challenging. We'll also lay some groundwork with a simple working example of our first two primary tasks as well as get a preview of the application you'll build at the end of this book: a fact-based question answering system. With that, let's look at some of the motivation for taming text by scoping out the size and shape of the information world we live in.

1.1 Why taming text is important

Just for fun, try to imagine going a whole day without reading a single word. That's right, one whole day without reading any news, signs, websites, or even watching television. Think you could do it? Not likely, unless you sleep the whole day. Now spend a moment thinking about all the things that go into reading all that content: years of schooling and hands-on feedback from parents, teachers, and peers; and countless spelling tests, grammar lessons, and book reports, not to mention the hundreds of thousands of dollars it takes to educate a person through college. Next, step back another level and think about how much content you *do* read in a day.

To get started, take a moment to consider the following questions:

- How many email messages did you get today (both work and personal, including spam)?
- How many of those did you read?
- How many did you respond to right away? Within the hour? Day? Week?
- How do you find old email?
- How many blogs did you read today?
- How many online news sites did you visit?

- Did you use instant messaging (IM), Twitter, or Facebook with friends or colleagues?
- How many searches did you do on Google, Yahoo!, or Bing?
- What documents on your computer did you read? What format were they in (Word, PDF, text)?
- How often do you search for something locally (either on your machine or your corporate intranet)?
- How much content did you produce in the form of emails, reports, and so on?

Finally, the big question: how much time did you spend doing this?

If you're anything like the typical information worker, then you can most likely relate to IDC's (International Data Corporation) findings from their 2009 study (Feldman 2009):

> Email consumes an average of 13 hours per week per worker... But email is no longer the only communication vehicle. Social networks, instant messaging, Yammer, Twitter, Facebook, and LinkedIn have added new communication channels that can sap concentrated productivity time from the information worker's day. The time spent searching for information this year averaged 8.8 hours per week, for a cost of $14,209 per worker per year. Analyzing information soaked up an additional 8.1 hours, costing the organization $13,078 annually, making these two tasks relatively straightforward candidates for better automation. It makes sense that if workers are spending over a third of their time searching for information and another quarter analyzing it, this time must be as productive as possible.

Furthermore, this survey doesn't even account for how much time these same employees spend creating content during their personal time. In fact, eMarketer estimates that internet users average 18 hours a week online (eMarketer) and compares this to other leisure activities like watching television, which is still king at 30 hours per week.

Whether it's reading email, searching Google, reading a book, or logging into Facebook, the written word is everywhere in our lives.

We've seen the individual part of the content picture, but what about the collective picture? According to IDC (2011), the world generated *1.8 zettabytes* of digital information in 2011 and "by 2020 the world will generate 50 times [that amount]." Naturally, such prognostications often prove to be low given we can't predict the next big trend that will produce more content than expected.

Even if a good-size chunk of this data is due to signal data, images, audio, and video, the current best approach to making all this data findable is to write analysis reports, add keyword tags and text descriptions, or transcribe the audio using speech recognition or a manual closed-captioning approach so that it can be treated as text. In other words, no matter how much structure we add, it still comes back to text for us to share and comprehend our content. As you can see, the sheer volume of content can be daunting, never mind that text processing is also a hard problem on a small scale, as you'll see in a later section. In the meantime, it's worthwhile to think about what the ideal applications or tools would do to help stem the tide of text that's

engulfing us. For many, the answer lies in the ability to quickly and efficiently hone in on the answer to our questions, not just a list of possible answers that we need to then sift through. Moreover, we wouldn't need to jump through hoops to ask our questions; we'd just be able to use our own words or voice to express them with no need for things like quotations, AND/OR operators, or other things that make it easier on the machine but harder on the person.

Though we all know we don't live in an ideal world, one of the promising approaches for taming text, popularized by IBM's Jeopardy!-playing Watson program and Apple's Siri application, is a question answering system that can process natural languages such as English and return *actual* answers, not just pages of *possible* answers. In *Taming Text*, we aim to lay some of the groundwork for building such a system. To do this, let's consider what such a system might look like; then, let's take a look at some simple code that can find and extract key bits of information out of text that will later prove to be useful in our QA system. We'll finish off this chapter by delving deeper into why building such a system as well as other language-based applications is so hard, along with a look at how the chapters to follow in this book will lay the foundation for a fact-based QA system along with other text-based systems.

1.2 Preview: A fact-based question answering system

For the purposes of this book, a QA system should be capable of ingesting a collection of documents suspected to have answers to questions that users might ask. For instance, Wikipedia or a collection of research papers might be used as a source for finding answers. In other words, the QA system we propose is based on identifying and analyzing text that has a chance of providing the answer based on patterns it has seen in the past. It won't be capable of inferring an answer from a variety of sources. For instance, if the system is asked "Who is Bob's uncle?" and there's a document in the collection with the sentences "Bob's father is Ola. Ola's brother is Paul," the system wouldn't be able to infer that Bob's uncle is Paul. But if there's a sentence that directly states "Bob's uncle is Paul," you'd expect the system to be able to answer the question. This isn't to say that the former example can't be attempted; it's just beyond the scope of this book.

A simple workflow for building the QA system described earlier is outlined in figure 1.1.

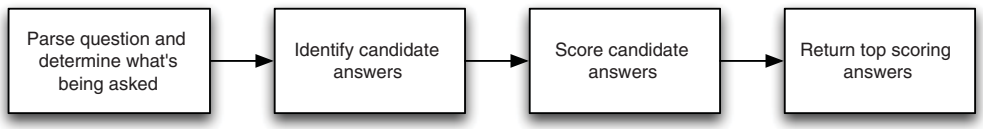

Figure 1.1 A simple workflow for answering questions posed to a QA system

Naturally, such a simple workflow hides a lot of details, and it also doesn't cover the ingestion of the documents, but it does allow us to highlight some of the key components needed to process users' questions. First, the ability to parse a user's question and determine what's being asked typically requires basic functionality like identifying words, as well as the ability to understand what kind of answer is appropriate for a question. For instance, the answer to "Who is Bob's uncle?" should likely be a person, whereas the answer to "Where is Buffalo?" probably requires a place-name to be returned. Second, the need to identify candidate answers typically involves the ability to quickly look up phrases, sentences, or passages that contain potential answers without having to force the system to parse large quantities of text.

Scoring implies many of the basic things again, such as parsing words, as well as a deeper understanding of whether a candidate actually contains the necessary components to answer a question, such as mentioning a person or a place. As easy as some of these things sound given the ease with which most humans *think* they do these things, they're not to be taken for granted. With this in mind, let's take a look at an example of processing a chunk of text to find passages and identify interesting things like names.

1.2.1 Hello, Dr. Frankenstein

In light of our discussion of a question answering system as well as our three primary tasks for working with text, let's take a look at some basic text processing. Naturally, we need some sample text to process in this simple system. For that, we chose Mary Shelley's classic *Frankenstein*. Why *Frankenstein*? Besides the authors' liking the book from a literary standpoint, it also happens to be the first book we came across on the Gutenberg Project site (http://www.gutenberg.org/), it's plain text and nicely formatted (which you'll find is a rarity in your day-to-day life with text), and there's the added bonus that it's out of copyright and freely distributable. We've included a full copy in our source tree, but you can also download a copy of the book at http://www.gutenberg.org/cache/epub/84/pg84.txt.

Now that we have some text to work with, let's do a few tasks that come up time and time again in text applications:

- Search the text based on user input and return the relevant passage (a paragraph in this example)
- Split the passage into sentences
- Extract "interesting" things from the text, like the names of people

To accomplish these tasks, we'll use two Java libraries, Apache Lucene and Apache OpenNLP, along with the code in the com.tamingtext.frankenstein.Frankenstein Java file that's included with the book and also available on GitHub at http://www.github.com/tamingtext/book. See https://github.com/tamingtext/book/blob/master/README for instructions on building the source.

The high-level code that drives this process can be seen in the following listing.

Listing 1.1 Frankenstein driver program

```
Frankenstein frankenstein = new Frankenstein();
frankenstein.init();
frankenstein.index();
String query = null;
while (true) {
  query = getQuery();
  if (query != null) {
    Results results = frankenstein.search(query);
    frankenstein.examineResults(results);
    displayResults(results);
  } else {
    break;
  }
}
```

Make content searchable.

Prompt user for query.

Perform search.

Parse results and show interesting items.

In the driver example, you first index the content. Indexing is the process of making the content searchable using Lucene. We'll explain this in more detail in the chapter on search later in the book. For now, you can think of it as a quick way of looking up where words occur in a piece of text. Next you enter a loop where you ask the user to enter a query, execute the search, and then process the discovered results. For the purposes of this example, you treat each paragraph as a searchable unit. This means that when you execute a search, you'll be able to know exactly which paragraph in the book matched the query.

After you have your paragraphs, you switch over to using OpenNLP, which will take each paragraph, split it into sentences, and then try to identify the names of people in a sentence. We'll forgo examining the details of how each of the methods are implemented, as various sections in the remainder of the book cover the concepts. Instead, let's run the program and try a query and look at the results.

To run the code, open a terminal window (command prompt and change into the directory containing the unpacked source code) and type `bin/frankenstein.sh` on UNIX/Mac or `bin/frankenstein.cmd` on Windows. You should see the following:

```
Initializing Frankenstein
Indexing Frankenstein
Processed 7254 lines.  Paragraphs: 722

Type your query.  Hit Enter to process the query \
(the empty string will exit the program):
>
```

At this point, you can enter a query, such as `"three months"`. A partial listing of the results follows. Note that we've inserted [...] in numerous places for formatting purposes.

```
>"three months"
Searching for: "three months"
Found 4 total hits.
-----------------------------------
Match: [0] Paragraph: 418
Lines: 4249-4255
```

```
    "'Do you consider,' said his companion to him, ...
    ----- Sentences ----
        [0] "'Do you consider,' said his companion to him, ...
        [1] I do not wish to take any unfair advantage, ...
-----------------------------------
Match: [1] Paragraph: 583
Lines: 5796-5807
    The season of the assizes approached.  ...
    ----- Sentences ----
...     [2] Mr. Kirwin charged himself with every care ...
         >>>> Names
             Kirwin
...     [4] ... that I was on the Orkney Islands ...
         >>>> Locations
             Orkney Islands

-----------------------------------
Match: [2] Paragraph: 203
Lines: 2167-2186
    Six years had elapsed, passed in a dream but for one indelible trac
e, ...
    ----- Sentences ----
... [4] ... and represented Caroline Beaufort in an ...
        >>>> Names
            Caroline Beaufort

...     [7] While I was thus engaged, Ernest entered:  ...  "Welcome
, my dearest Victor," said he.  "Ah!
        >>>> Names
            Ah

        [8] I wish you had come three months ago, and then you would ha
ve found us all joyous and delighted.
        >>>> Dates
            three months ago

        [9] ... who seems sinking under his misfortune; and your pers
uasions will induce poor Elizabeth to cease her ...
        >>>> Names
            Elizabeth

        ...
```

This output shows the results of fetching the top paragraphs that mention "three months" as a phrase (four paragraphs in all) along with a few examples of the sentences in the paragraph, as well as lists of any names, dates, or locations in that text. In this example, you can see samples of the sentence detection as well as the extraction of names, locations, and dates. A keen eye will also notice a few places where the simple system is clearly wrong. For instance, the system thinks *Ah* is a name, but that *Ernest* isn't. It also failed to split the text ending in "... said he. "Ah!" into separate sentences. Perhaps our system doesn't know how to properly handle exclamation points or there was some odd formatting in the text.

For now, we'll wave our hands as to why these failed. If you explore further with other queries, you'll likely find plenty of the good, bad, and even the ugly in processing text. This example makes for a nice segue into our next section, which will touch

on some of these difficulties in processing text as well as serve as motivation for many of the approaches we take in the book.

1.3 Understanding text is hard

Suppose Robin and Joe are talking, and Joe states, "The bank on the left is solid, but the one on the right is crumbling." What are Robin and Joe talking about? Are they on Wall Street looking at the offices of two financial institutions, or are they floating down the Mississippi River looking for a place to land their canoe? If you assume the former, the words *solid* and *crumbling* probably refer to the state of the banks' finances, whereas the latter case is an assessment of the quality of the ground on the side of a river. Now, what if you replaced the characters' names with the names *Huck* and *Tom* from *The Adventures of Tom Sawyer*? You'd likely feel pretty confident in stating it's a river bank and not a financial institution, right? As you can see, context is also important. It's often the case that only with more information from the surrounding context combined with your own experiences can you truly know what some piece of content is about. The ambiguity in Joe's statement only touches on the surface of the complexity involved in understanding text.

Given well-written, coherent sentences and paragraphs, knowledgeable people seamlessly look up the meanings of words and incorporate their experiences and knowledge of their surroundings to arrive at an understanding of content and conversations. Literate adults can (more or less) effortlessly dissect sentences, identify relationships, and infer meaning nearly instantaneously. And, as in the Robin and Joe example, people are almost always aware when something is significantly out of place or lacking from a sentence, paragraph, or document as a whole. Human beings also feed off others in conversation, instantly adapting tone and emotions to convey thoughts on subjects ranging from the weather to politics to the role of the designated hitter. Though we often take these skills for granted, we should remember that they have been fine-tuned through many years of conversation, education, and feedback from others, not to mention all the knowledge passed down from our ancestors.

At the same time, computers and the fields of information retrieval (IR) and natural language processing (NLP) are still relatively young. Computers need to be capable of processing language on many different levels in order to come close to "understanding" content like people do. (For an in-depth discussion of the many factors that go into NLP, see Liddy [2001].) Though full understanding is a tall order for a computer, even doing basic tasks can be overwhelming given the sheer volume of text available and the variety with which it occurs.

There's a reason the saying goes "the numbers don't lie" and not "the text doesn't lie"; text comes in all shapes and meanings and trips up even the smartest people on a regular basis. Writing applications to process text can mean facing a number of technical and nontechnical challenges. Table 1.2 outlines some of the challenges text applications face, each row increasing in difficulty from the previous.

Table 1.1 Processing text presents challenges at many levels, from handling character encodings to inferring meaning in the context of the world around us.

Level	Challenges
Character	– Character encodings, such as ASCII, Shift-JIS, Big 5, Latin-1, UTF-8, UTF-16. – Case (upper and lower), punctuation, accents, and numbers all require different treatments in different applications.
Words and morphemes[a]	– Word segmentation: dividing text into words. Fairly easy for English and other languages that use whitespace; much harder for languages like Chinese and Japanese. – Assigning part of speech. – Identifying synonyms; synonyms are useful for searching. – Stemming: the process of shortening a word to its base or root form. For example, a simple stemming of *words* is *word*. – Abbreviations, acronyms, and spelling also play important roles in understanding words.
Multiword and sentence	– Phrase detection: *quick red fox*, *hockey legend Bobby Orr*, and *big brown shoe* are all examples of phrases. – Parsing: breaking sentences down into subject-verb and other relationships often yields useful information about words and their relationships to each other. – Sentence boundary detection is a well-understood problem in English, but is still not perfect. – Coreference resolution: "Jason likes dogs, but he would never buy one." In this example, *he* is a coreference to Jason. The need for coreference resolution can also span sentences. – Words often have multiple meanings; using the context of a sentence or more may help choose the correct word. This process is called *word sense disambiguation* and is difficult to do well. – Combining the definitions of words and their relationships to each other to determine the meaning of a sentence.
Multisentence and paragraph	At this level, processing becomes more difficult in an effort to find deeper understanding of an author's intent. Algorithms for summarization often require being able to identify which sentences are more important than others.
Document	Similar to the paragraph level, understanding the meaning of a document often requires knowledge that goes beyond what's contained in the actual document. Authors often expect readers to have a certain background or possess certain reading skills. For example, most of this book won't make much sense if you've never used a computer and done some programming, whereas most newspapers assume at least a sixth-grade reading level.
Multidocument and corpus	At this level, people want to quickly find items of interest as well as group related documents and read summaries of those documents. Applications that can aggregate and organize facts and opinions and find relationships are particularly useful.

a. A *morpheme* is a small linguistic unit that still has meaning. Prefixes and suffixes are examples of morphemes.

Beyond these challenges, human factors also play a role in working with text. Different cultures, different languages, and different interpretations of the same writing can leave even the best engineer wondering what to implement. Merely looking at some sample files and trying to extrapolate an approach for a whole collection of documents is often problematic. On the other side of the coin, manually analyzing and annotating large sets of documents can be expensive and time consuming. But rest assured that help is available and text can be tamed.

1.4 Text, tamed

Now that you've seen some of the challenges you're about to face, take heart knowing that many tools exist both commercially and in the open source community (see http://www.opensource.org) to tackle these topics and many more. One of the great things about the journey you're embarking on is its ever-changing and ever-improving nature. Problems that were intractable 10 years ago due to resource limits are now yielding positive results thanks to better algorithms, faster CPUs, cheaper memory, cheaper disk space, and tools for easily harnessing many computers into a single virtual CPU. Now, more than ever, quality open source tools exist that can form the foundation for new ideas and new applications.

This book is written to bring real-world experience to these open source tools and introduce you to the fields of natural language processing and information retrieval. We can't possibly cover all aspects of NLP and IR nor are we going to discuss bleeding-edge research, at least not until the end of the book; instead we'll focus on areas that are likely to have the biggest impact in taming your text.

By focusing on topics like search, *entity identification* (finding people, places, and things), grouping and labeling, clustering, and summarization, we can build practical applications that help users find and understand the important parts of their text quickly and easily.

Though we hate to be a buzzkill on all the excitement of taming text, it's important to note that there are no perfect approaches in working with text. Many times, two people reviewing the same output won't agree on the correctness of the results, nor will it be obvious what to fix to satisfy them. Furthermore, fixing one problem may expose other problems. Testing and analysis are as important as ever to achieving quality results. Ultimately, the best systems take a human-in-the-loop approach and learn from user feedback where possible, just as smart people learn from their mistakes and from their peers. The user feedback need not be explicit, either. Capturing clicks, and analyzing logs and other user behaviors can provide valuable feedback on how your users are utilizing your application. With that in mind, here are some general tips for improving your application and keeping your sanity:

- Get to know your users. Do they care about certain structures like tables and lists, or is it enough to collect all the words in a document? Are they willing to give you more information in return for better results, or is simplicity the rule? Are they willing to wait longer for better results, or do they need a best guess immediately?

- Get to know your content. What file formats (HTML, Microsoft Word, PDF, text) are used? What structures and features are important? Does the text contain a lot of jargon, abbreviations, or different ways of saying the same thing? Is the content focused on a single area of interest or does it cover a number of topics?
- Test, test, and test some more. Take the time (but not too much time) to measure the quality of your results and the cost of obtaining them. Become practiced in the art of arbitration. Every nontrivial text-based application will need to make trade-offs in regards to quality and scalability. By combining your knowledge of your users and your content, you can often find the sweet spot of quality and performance that satisfies most people most of the time.
- Sometimes, a best guess is as good as it gets. Look for ways to provide confidence levels to your users so they can make an informed decision about your response.
- All else being equal, favor the simpler approach. Moreover, you'll be amazed at how good simple solutions can be at getting decent results.

Also, though working in non-native languages is an interesting problem in itself, we'll stick to English for this book. Rest assured that many of the approaches can be applied to other languages given the right resources.

It should also be pointed out that the kinds of problems you might wish to solve range in difficulty from relatively straightforward to so hard you might as well flip a coin. For instance, in English and other European languages, tokenization and part of speech tagging algorithms perform well, whereas tools like machine translation of foreign languages, sentiment analysis, and reasoning from text are much more difficult and often don't perform well in unconstrained environments.

Finally, text processing is much like riding a roller coaster. There will be highs when your application can do no wrong and lows when your application can do no right. The fact is that none of the approaches discussed in this book or in the broader field of NLP are the final solution to the problem. Therein lies the ultimate opportunity for you to dig in and add your signature. So let's get started and lay the foundation for the ideas to come in later chapters by setting the context that takes us beyond search into the wonderful world of natural language processing.

1.5 Text and the intelligent app: search and beyond

For many years now, search has been king. Without the likes of Google and Yahoo!, there's no doubt that the internet wouldn't be anywhere near what it is today. Yet, with the rise of good open source search tools like Apache Solr and Apache Lucene, along with a myriad of crawlers and distributed processing techniques, search is a commodity, at least on the smaller scale of personal and corporate search where huge data centers aren't required. At the same time, people's expectations of search engines are increasing. We want better results in less time while entering only one or two keywords. We also want our own content easily searched and organized.

Furthermore, corporations are under huge pressure to constantly add value. Every time some big player like Google or Amazon makes a move to better access information, the bar is raised for the rest of us. Five, ten, or fifteen years ago, it was enough to add search capabilities to be able to find data; now search is a prerequisite and the game-changing players use complex algorithms utilizing machine learning and deep statistical analysis to work with volumes of data that would take people years to understand. This is the evolution of the intelligent application. More and more companies are adopting machine learning and deep text analysis in well-defined areas to bring more intelligence to their applications.

The adoption of machine learning and NLP techniques is grounded in the reality of practical applications dealing with large volumes of data, and not the grandiose, albeit worthwhile, notion of machines "understanding" people or somehow passing the Turing Test (see http://en.wikipedia.org/wiki/Turing_Test). These companies are focused on finding and extracting important text features; aggregating information like user clicks, ratings, and reviews; grouping and summarizing similar content; and, finally, displaying all of these features in ways that allow end users to better find and use the content, which should ultimately lead to more purchases or traffic or whatever is the objective. After all, you can't buy something if you can't find it, right?

So, how do you get started doing all of these great things? You start by establishing the baseline with search (covered in chapter 3) and then examine ways of automatically organizing content using concepts that you employ in your daily life. Instead of doing it manually, you let the machine do it for you (with a little help when needed). With that in mind, the next few sections break down the ideas of search and organizing content into three distinct areas and propose an example that ties many of the concepts together, which will be explored more completely in the ensuing chapters.

1.5.1 Searching and matching

Search provides the starting point for most of your text taming activities, including our proposed QA system, where you'll rely on it both for indexing the input data as well as for identifying candidate passages that match a user's question. Even when you need to apply techniques that go beyond search, you'll likely use search to find the subset of text or documents on which to apply more advanced techniques.

In chapter 3, "Searching," we'll explore how to make documents available for searching, indexing, and how to retrieve documents based on a query. We'll also explore how documents are ranked by a search engine and use this information to improve the returned results. Finally, we'll examine faceted search, which allows searches to be refined by limiting results to a predefined category. The coverage of these topics will be grounded in examples using Apache Solr and Apache Lucene.

After you're familiar with the techniques of search, you'll quickly realize that search is only as good as the content backing that search. If the words and phrases that your users are looking for aren't in your index, then you won't be able to return a relevant result. In chapter 4, "Fuzzy string matching," we'll look at techniques for

enabling query recommendations based on the content that's available via query spell-checking as well as how these same techniques can be applied to database- or record-linking tasks that go beyond simple database joins. These techniques are often used not only as part of search, but also for more complex things like identifying whether two user profiles are the same person, as might happen when two companies merge and their customer lists must be combined.

1.5.2 Extracting information

Though search will help you find documents that contain the information you need, often you need to be able to identify smaller units of information. For instance, the ability to identify proper names in a large collection of text can be immensely helpful in tracking down criminal activity or finding relationships between people who might not otherwise meet. To do this we'll explore techniques for identifying and classifying small selections of text, typically just a few words in length.

In chapter 2, "Foundations of taming text," we'll introduce techniques for identifying words that form a linguistic unit such as noun phrases, which can be used to identify words in a document or query which can be grouped together. In chapter 5, "Identifying people, places, and things," we'll look at how to identify proper names and numeric phrases and put them into semantic categories such as person, location, and date, irrespective of their linguistic usage. This ability will be fundamental to your ability to build a QA system in chapter 8. For both of these tasks we'll use the capabilities of OpenNLP and explore how to use its existing models as well as build new models that better fit the data. Unlike the problem of searching and matching, these models will be built from examining manually annotated content and then using statistical machine learning approaches to produce a model.

1.5.3 Grouping information

The flip side to extracting information from text is adding supplemental information to your text by grouping it together or adding labels. For example, think about how much easier it would be to process your email if it were automatically tagged and prioritized so that you could also find all emails that are similar to one another. This way, you could focus in on just those emails that require your immediate attention as well as find supporting content for emails you're sending.

One common approach to this is to group your text into categories. As it turns out, the techniques used for extracting information can also be applied to grouping text or documents into categories. These groups can often then be used as facets in your search index, supplemental keywords, or as an alternate way for users to navigate information. Even in cases where your users are providing the categories via tagging, these techniques can recommend tags that have been used in the past. Chapter 7, "Classification, categorization, and tagging," shows how to build models to classify documents and how to apply these models to new documents to improve user experience with text.

When you've tamed your text and are able to find what you're looking for, and you've extracted the information needed, you may find you have too much of a good thing. In chapter 6, "Clustering text," we'll look at how to group similar information. These techniques can be used to identify redundant information and, if necessary, suppress it. They can also be used to group similar documents so that a user can peruse entire topics at a time and access the relevancy of multiple documents at once without having to read each document.

1.5.4 An intelligent application

In our penultimate chapter, "Building an example question answering system," we'll bring a number of the approaches described in the early chapters together to build an intelligent application. Specifically, you'll build a fact-based question answering system designed to find answers to trivia-like questions in text. For instance, given the right content, you should be able to answer questions like, "Who is the President of the United States?" This system uses the techniques of chapter 3, "Searching," to identify text that might contain the answer to your question. The approaches presented in chapter 5, "Identifying people, places, and things," will be used to find these pieces of text that are often the answers to fact-based questions. The material in chapter 2, "Foundations of taming text," and chapter 7, "Classification, categorization, and tagging," will be used to analyze the question being asked, and determine what type of information the question is looking for. Finally, you'll apply the techniques for document ranking described in chapter 3 to rank your answers.

1.6 Summary

Taming text is a large and sometimes overwhelming task, further complicated by different languages, different dialects, and different interpretations. Text can appear as elegant prose written by great authors or the ugliest of essays written without style or substance. Whatever its form, text is everywhere and it must be dealt with by people and programs. Luckily, many tools exist both commercially and in open source to help try to make sense of it all. It won't be perfect, but it's getting better all the time. So far, we've taken a look at some of the reasons why text is so important as well as hard to process. We've also looked at what role text plays in the intelligent web, introduced the topics we'll cover, and gave a brief overview of some of the things needed to build a simple question answering system. In the next chapter, we'll kick things off by laying down the foundations of text analysis along with some basics on extracting raw text from the many file formats found in the wild today.

1.7 Resources

"Americans Spend Half of Their Spare Time Online." 2007. Media-Screen LLC. http://www.media-screen.com/press050707.html.

Feldman, Susan. 2009. "Hidden Costs of Information Work: A Progress Report." International Data Corporation.

Gantz, John F. and Reinsel, David. 2011. "Extracting Value from Chaos." International Data Corporation. http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf.

Liddy, Elizabeth. 2001. "Natural Language Processing." *Encyclopedia of Library and Information Science, 2nd Ed.* NY. Marcel Decker, Inc.

"Trends in Consumers' Time Spent with Media." 2010. eMarketer. http://www.emarketer.com/Article.aspx?R=1008138.

JAVA/MACHINE LEARNING

Taming Text

Ingersoll • Morton • Farris

There is so much information in our lives, we are practically drowning in it. Fortunately, there are practical tools and techniques for managing unstructured text that can throw the smart developer a much-needed lifeline. You'll find them in this book.

Taming Text is a practical, example-driven guide to working with text in real applications. This book introduces you to useful techniques like full-text search, proper name recognition, clustering, tagging, information extraction, and summarization. You'll explore real use cases as you systematically absorb the foundations upon which they are built.

What's Inside

- Text-taming techniques
- Libraries like Solr and Mahout
- How to build text-processing applications

This book avoids jargon, presenting the subject clearly and concisely, so you can understand it without a background in statistics or natural language processing. Examples are in Java, but the concepts can be applied in any language.

Grant Ingersoll is an engineer, speaker, and trainer, a Lucene committer, and a cofounder of the Mahout machine-learning project. **Thomas Morton** is the primary developer of OpenNLP and Maximum Entropy. **Drew Farris** is a technology consultant, software developer, and contributor to Mahout, Lucene, and Solr.

"Takes the mystery out of very complex processes."
—From the Foreword by Liz Liddy Dean, iSchool, Syracuse University

"Text analysis and processing as it should be: clear, practical, and open source!"
—David Weiss, Carrot Search s.c.

"Shows how to unlock and exploit information locked up in text documents."
—Rick Wagner, Red Hat

"Teaches text concepts with examples ... makes text search easy."
—Doug Warren, Java Web Services

"A great overview of tools and techniques for text processing."
—Julien Nioche, DigitalPebble, Ltd.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/TamingText

ISBN-13: 978-1-933988-38-2
ISBN-10: 1-933988-38-X
54499
9 781933 988382

MANNING US $44.99 / Can $47.99 [including eBook]