CHAPTER 9

Text Mining

In this chapter we will learn how to extract patterns and discover new knowledge by applying many of the techniques we have learned so far, not on ordered data, but on unstructured natural language. This constitutes the vast and fast growing area of text and web mining. For all the techniques described up to this point, a cleaned and organized table consisting of rows and columns of data was fed as input to an algorithm. The output from the algorithm was a model that could then be used to predict outcomes from a new data set or to find patterns in data. But how do we apply the same techniques to extract patterns and predict outcomes when the input data looks like normal written communication or worse? This might seem baffling at first, but as we shall see in this chapter, there are ways of presenting text data to the algorithms that process "normal" data.

We start out with a brief historical introduction to the field of text mining to establish some context. In the following section, we will describe techniques that can convert common text into a semi-structured format that we, and the algorithms we have introduced so far, can recognize. Finally we will introduce the toolkit that is available in RapidMiner to do this conversion and apply it in two case studies: one involving an unsupervised (clustering) model and another involving a supervised (SVM) model. We will close the chapter with some key considerations to keep in mind while implementing text mining.

Text mining is the new frontier of predictive analytics and data mining. Eric Siegel in his book *Predictive Analytics* (Siegel, 2013) provides an interesting analogy: if all the data in the world was equivalent to the water on earth, then textual data is like the ocean, making up a majority of the volume. Text analytics is driven by the need to process natural human language, but unlike numeric or categorical data, natural language does not exist in a "structured" format consisting of rows (of examples) and columns (of attributes). Text mining is therefore the domain of unstructured data mining.

Some of the first applications of text mining came about when people were trying to organize documents (Cutting, 1992). Hearst (Hearst, June 20-26, 1999)

Predictive Analytics and Data Mining. http://dx.doi.org/10.1016/B978-0-12-801460-8.00009-4

IT IS NLP, MY DEAR WATSON!

Perhaps the most famous application of text mining is IBM's Watson program, which performed spectacularly when competing against humans on the nightly game show Jeopardy! How does Watson use text mining? Watson has instant access to hundreds of millions of structured and unstructured documents, including the full content of Wikipedia entries.

When a Jeopardy! question is transcribed to Watson, it searches for and identifies candidate documents that score a very close match to the words of the question. The search and comparison methods it uses are similar to those used by search engines, and include many of the techniques, such as n-grams and stemming, which we discuss in this chapter. Once it identifies candidate documents, it again uses other text mining (also known as natural language processing or NLP) methods to rank them. For example, if the answer is, REGARDING THIS DEVICE, ARCHIMEDES SAID, "GIVE ME A PLACE TO STAND ON, AND I WILL MOVE THE EARTH, a Watson search for this sentence in its databases might reveal among its candidate documents several with the term "lever." Watson might insert the word "lever" inside the answer text and rerun a new search to see if there are other documents with the new combination of terms. If the search result has many matches to the terms in the sentence—as it most likely would in this case—a high score is assigned to the inserted term.

If a broad and non-domain-focused program like Watson, which relies heavily on text mining and NLP, can answer open-ended quiz show questions with nearly 100% accuracy, one can imagine how successful specialized NLP tools would be. In fact IBM has successfully deployed a Watson-type program to help in decision making at health care centers (Upbin, 2013).

Text mining also finds applications in numerous business activities such as email spam filtering, consumer sentiment analysis, and patent mining to name a few. We will explore a couple of these in this chapter.

recognized that text analysis does not require artificial intelligence but "…a mixture of computationally-driven and user-guided analysis," which is at the heart of the supervised models used in predictive analytics that we have discussed so far.

People in the data warehousing and business intelligence domains can appreciate text mining in a slightly different context. Here, the objective is not so much discovering new trends or patterns, but cleaning data stored in business databases. For example, when people make manual entries into a customer relationship management (CRM) software, there is a lot of scope for typographic errors: a salesperson's name may be spelled "Osterman" in several instances (which is perhaps the correct spelling) and "Ostrerman" in a few instances, which is a misspelling. Text mining could be used in such situations to identify the "right" spelling and suggest it to the entry operator to ensure that data consistency is maintained. Similar application logic could be used in identifying and streamlining call center service data (McKnight, 2005).

Text mining, more than any other technique within data mining, fits the "mining" metaphor. Traditionally, mining refers to the process of separating dirt from valuable metal and in the case of text mining we attempt to separate valuable keywords from a mass of other words (or relevant documents from

a sea of documents) and use them to identify meaningful patterns or make predictions.

9.1 HOW TEXT MINING WORKS

The fundamental step in text mining involves converting text into semi-structured data. Once you convert the unstructured text into semi-structured data, there is nothing to stop you from applying any of the analytics techniques to classify, cluster, and predict. The unstructured text needs to be converted into a semi-structured dataset so that you can find patterns and even better, train models to detect patterns in new and unseen text. The chart in Figure 9.1 identifies the main steps in this process at a high level.

We will now examine each of the main processes in detail and introduce some terminology and concepts that are necessary. But before we describe these processes, we need to define a few core ideas that will be essential.

9.1.1 Term Frequency–Inverse Document Frequency (TF–IDF)

Consider a web search problem where the user types in some keywords and the search engine extracts all the documents (essentially, web pages) that contain

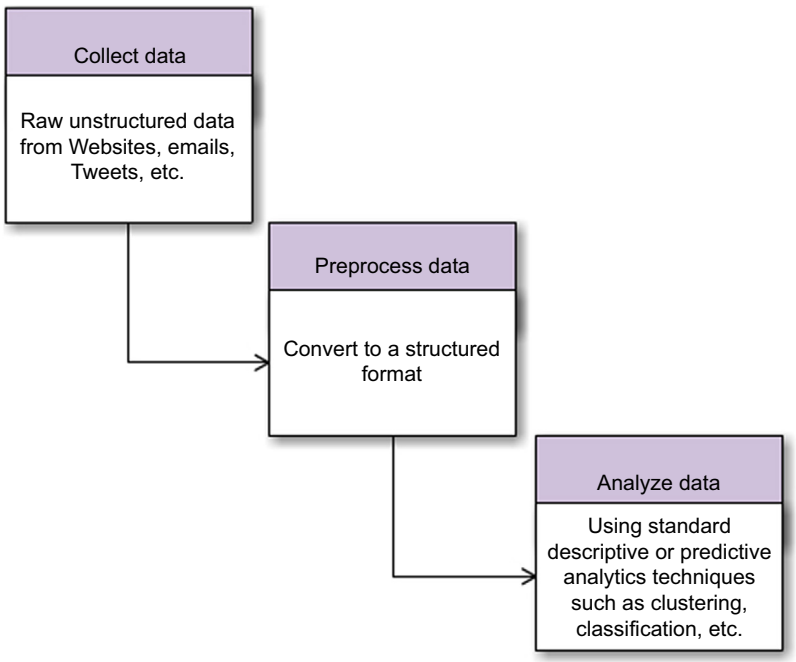

FIGURE 9.1
A high-level process for text mining.

these keywords. How does the search engine know which web pages to serve up? In addition to using network rank or page rank, the search engine also runs some form of text mining to identify the most relevant web pages. Suppose for example the user types in the following keywords: "RapidMiner books that describe text mining." In this case, the search engines run on the following basic logic:

1. Give a high weightage to those keywords that are relatively "rare."
2. Give a high weightage to those webpages that contain a large number of instances of the "rare" keywords.

In the above context, what is a "rare" keyword? Clearly, English words like "that," "books," "describe," and "text" possibly appear in a large number of web pages, whereas "RapidMiner" and "mining" may appear in a relatively smaller number of web pages. (A quick web search returned 382 million results for the word "books," whereas only 74,000 results were returned for "RapidMiner" at the time of this writing.) Therefore, these rarer keywords would receive a higher rating to begin with according to logic 1 above. Next, among all those pages that contain the rare keywords, only those pages that contain the largest number of instances of the rare keywords are likely to be the most relevant for the user and will receive high weightage according to logic 2 above. Thus the highest-weighted web pages are the ones for which the *product* of these two weights is the highest. Therefore, only those pages that not only contain the rare keywords, but have a high number of instances of the rare keywords should appear at the top of the search results.

The technique of calculating this weighting is called *TF–IDF*, which stands for Term Frequency–Inverse Document Frequency.

Calculating TF is very easy: it is simply the ratio of the number of times a keyword appears in a given document, $\mathbf{n_k}$ (where k is the keyword), to the total number of terms in the document, $\mathbf{n}$:

$$\mathrm{TF} = \mathbf{n}_k/\mathbf{n} \tag{9.1}$$

Considering the above example, a common English word such as "that" will have a fairly high TF score and a word such as "RapidMiner" will have a much lower TF score.

IDF is defined as follows:

$$\mathrm{IDF} = \log_2 (\mathbf{N}/\mathbf{N}_k) \tag{9.2}$$

where $\mathbf{N}$ is the number of documents under consideration (in a search engine context, $\mathbf{N}$ is the number of *ALL* the indexed webpages). For most text mining problems, $\mathbf{N}$ is the number of documents that we are trying to mine, and $\mathbf{N_k}$ is the number of documents thatcontain the keyword, k. Again, a word

such as "that" would arguably appear in every document and thus the ratio $(\mathbf{N}/\mathbf{N_k})$ would be close to 1, and the IDF score would be close to zero for "that." However, a word like "RapidMiner" would possibly appear in a relatively fewer number of documents and so the ratio $(\mathbf{N}/\mathbf{N_k})$ would be much greater than 1. Thus the IDF score would be high for this less common keyword.

Finally, TF–IDF is expressed as the simple product as shown below:

$$\mathrm{TF-IDF} = \mathrm{n_k}/\mathrm{n} * \log_2 (N/N_\mathrm{k}) \tag{9.3}$$

Going back to the above example, when we multiply the high TF for "that" by its corresponding very low IDF, we will get a very low (or zero) TF–IDF whereas when we multiply the low TF for "RapidMiner" by its corresponding fairly high IDF, we would get a relatively higher TF–IDF.

Typically, TF–IDF scores for every word in the set of documents is calculated in the preprocessing step of the three-step process described earlier. Performing this calculation will help in applying any of the standard data mining techniques that we discussed so far in this book. In the following sections we will describe additional concepts that are commonly employed in text mining.

9.1.2 Terminology

Consider the following two sentences: "This is a book on data mining" and "This book describes data mining and text mining using RapidMiner.. Let us suppose our objective is to perform a comparison between them, or a "similarity mapping." For our purpose, each sentence is one unit of text that needs to be analyzed.

These two sentences could be embedded in an email message or in two separate webpages or in two different text files or could be two sentences in the same text file. In the text mining context, each sentence is considered a distinct *document*. Furthermore, in the simplest case, words are separated by a special character: a blank space. Each word is called a *token*, and the process of discretizing words within a document is called *tokenization*. For our purpose here, each sentence can be considered a separate document, although what is considered an individual document may depend upon the context. For now, a document here is simply a sequential collection of tokens.

Document 1	This is a book on data mining.
Document 2	This book describes data mining and text mining using RapidMiner.

We can impose some form of structure on this raw data by creating a matrix where the columns consist of all the tokens found in the two documents and

the cells of the matrix are the counts of the number of times a token appears, as shown in Table 9.1.

Each token is now an attribute in standard data mining parlance and each document is an example. We therefore have a structured *example set*, to use standard terminology. Basically, unstructured raw data is now transformed into a format that is recognized, not only by the human users as a data table, but more importantly by all the machine learning algorithms which require such tables for training. This table is called a *document vector* or *term document matrix* (*TDM*) and is the cornerstone of the preprocessing required for text mining. Suppose we add a third statement, "RapidMiner is offered as an open source software program." This new document will increase the number of rows of our matrix by one (Document 3); however it increases the number of columns by seven (seven new words or tokens were introduced). This results in zeroes being recorded in nine other columns for row 3. As we add more new statements that have very little in common, we will end up with a very sparse matrix.

Note that we could have also chosen to use the term frequencies for each token instead of simply counting the number of occurrences and it would still be a sparse matrix. We can get TF by dividing each row of Table 9.1 by number of words in the row (document). This is shown in Table 9.2.[1]

Similarly, we could have also chosen to use the TF–IDF scores for each term to create the document vector. This is also shown in Figure 9.2.

One thing to notice in the two sample text documents above was the occurrence of common words such as "a," "this," "and," and other similar terms. Clearly in larger documents we would expect a larger number of such terms that do not really convey specific meaning. Most grammatical necessities such as articles, conjunctions, prepositions, and pronouns may need to be filtered before we perform additional analysis. Such terms are called *stopwords* and usually include most articles, conjunctions, pronouns, and prepositions. *Stopword filtering* is usually the second step that follows immediately after *tokenization*. Notice that our document vector has a significantly reduced size after applying standard English stopword filtering (see Figure 9.3).

In addition to filtering standard stopwords, we may also need to filter out some specific terms. For example, in analyzing text documents that pertain to the automotive industry, we may want to filter away terms that are common to this

[1]RapidMiner does a "double normalization" while calculating the TF scores. For example, in the case of Document 1, the TF score for the term "data" would be $(0.1428)/\sqrt{(0.1428^2 + 0.1428^2 + \ldots + 0.1428^2)}$ $= (0.1428)/\sqrt{7 * 0.1428^2} = 0.3779$ and so on for the other terms. This change in TF score calculation is reflected in the TF–IDF score. The double normalization makes it easy to apply algorithms such as SVM.

Table 9.1 Building a Matrix of Terms from Unstructured Raw Text

	this	is	a	book	on	data	mining	describes	text	rapidminer	and	using
Document 1	1	1	1	1	1	1	1	0	0	0	0	0
Document 2	1	0	0	1	0	1	2	1	1	1	1	1

Table 9.2 Using Term Frequencies Instead of Term Counts in a TDM

	this	is	a	book	on	data	mining	describes	text	rapidminer	and	Using
Document 1	1/7 = 0.1428	0.1428	0.1428	0.1428	0.1428	0.1428	0.1428	0	0	0	0	0
Document 2	1/10 = 0.1	0	0	0.1	0	0.1	0.2	0.1	0.1	0.1	0.1	0.1

ExampleSet (2 examples, 0 special attributes, 12 regular attributes)

Row No.	RapidMiner	This	a	and	book	data	describes	is	mining	on	text	using
1	0	0	0.577	0	0	0	0	0.577	0	0.577	0	0
2	0.447	0	0	0.447	0	0	0.447	0	0	0	0.447	0.447

FIGURE 9.2
Calculating TF–IDF scores for the sample TDM.

Row No.	RapidMiner	book	data	describes	mining	text	using
1	0	1	1	0	1	0	0
2	1	1	1	1	2	1	1

FIGURE 9.3
Stopword filtering reduces the size of the TDM significantly.

industry such as "car," "automobile," "vehicle," and so on. This is generally achieved by creating a separate dictionary where we define these context-specific terms and then apply *term filtering* to remove them from our data. (*Lexical substitution* is the process of finding an alternative for a word in the context of a clause and is used to align all the terms to the same term based upon the field or subject which is being analyzed—this is especially important in areas with specific jargon, for example, in clinical settings.)

We may encounter words such as "recognized," "recognizable," or "recognition" in different usages, but contextually they may all imply the same meaning. For example, "Einstein is a *well-recognized* name in physics" or "The physicist went by the easily *recognizable* name of Einstein" or "Few other physicists have the kind of name *recognition* that Einstein has." The so-called root of all these highlighted words is "recognize." By reducing terms in a document to their basic stems, we can simplify the conversion of unstructured text to structured data because we now only take into account the occurrence of the root terms. This process is called *stemming*. The most common stemming technique for text mining in English is the Porter method (Porter, 1980). Porter stemming works on a bunch of rules where the basic idea is to remove and/or replace the suffix of words. For example, one rule would be "Replace all terms which end in 'ies' by 'y,'" such as replacing the term "anomal*ies*" with "anomol*y*". Similarly, another rule would be to stem all terms ending in "s" by removing the "s," as in "algorithms" to "algorithm." While the Porter stemmer is very efficient, it can make mistakes that could prove costly. For example, "arms" and "army" would both be stemmed to "arm," which would result in somewhat different contextual meaning. There are other stemmers available; which one you choose is usually guided by experience in your domain. Stemming is usually the next process step following term filtering. (A word of caution: stemming is completely dependent upon the human language being processed as well as the period of the language being processed.

Row...	label	RapidMiner	book	book_data	book_descr...	data	data_mining	describes	describes_data	mining	mining_text	mining_usi...	text_0	text_mining	using	using_RapidMiner
1	text1	0	0.447	0.447	0	0.447	0.447	0	0	0.447	0	0	0	0	0	0
2	text2	0.243	0.243	0	0.243	0.243	0.243	0.243	0.243	0.485	0.243	0.243	0.243	0.243	0.243	0.243

FIGURE 9.4
Meaningful n-grams show higher TF–IDF scores.

Table 9.3 A Typical Sequence of Preprocessing Steps to Use in Text Mining

Step	Action	Result
1	Tokenize	Convert each word or term in a document into a distinct attribute
2	Stopword removal	Remove highly common grammatical tokens/words
3	Filtering	Remove other very common tokens
4	Stemming	Trim each token to its most essential minimum
5	n-grams	Combine commonly occurring token pairs or tuples (more than 2)

Historical usage varies so widely that comparing text across generations—Shakespeare to present-day literature for instance—can raise concerns.)

There are families of words in the spoken and written language that typically go together. For example, the word "Good" is usually followed by either "Morning," "Afternoon," "Evening," "Night," or in Australia, "Day." Grouping such terms, called *n-grams*, and analyzing them statistically can present new insights. Search engines use word n-gram models for a variety of applications, such as automatic translation, identifying speech patterns, checking misspelling, entity detection, information extraction, among many different use cases. Google has processed more than a trillion words (1,024,908,267,229 words back as far back as 2006) of running text and has published the counts for all 1,176,470,663 five-word sequences that appear at least 40 times (Franz, 2006). While most text mining applications do not require 5-grams, bigrams and trigrams are quite useful. The final preprocessing step typically involves forming these n-grams and storing them in our document vector. Also, most algorithms providing n-grams become computationally expensive and the results become huge so in practice the amount of "n" will vary based upon the size of the documents and the corpus.

Figure 9.4 shows a TF-based document vector for bigrams (n = 2) from our examples and as you can see, terms like "data mining" and "text mining" and "using RapidMiner" can be quite meaningful in this context. Table 9.3 summarizes a typical sequence of preprocessing steps that will convert unstructured data into a semi-structured format.

Usually there is a preprocessing step before tokenization such as removing special characters, changing the case (upcasing and downcasing), or sometimes even performing a simple spell check beforehand. Data quality in text mining is just as important as in other areas.

9.2 IMPLEMENTING TEXT MINING WITH CLUSTERING AND CLASSIFICATION

We have introduced a few essential concepts that would be needed for a basic text mining project. In the following sections, we will examine two case studies that apply text mining. In the first example, we will take several documents (web pages) and group keywords found in them into similar *clusters*. In the second example, we will attempt to perform a *blog gender classification*. We start with several blogs (documents) written by men and women authors, to be used as training data. Using the article keywords as features, we will train several classification models, including a couple of SVMs, to recognize stylistic characteristics of authors and classify new unseen blogs as belonging to one of the two author classes (male or female).

9.2.1 Case Study 1: Keyword Clustering

In this first example, we will introduce some of the web mining features of RapidMiner and then create a clustering model with keyword data mined from a website. The objective of this case is to scan several pages from a given website and identify the most frequent words within these pages that also serve to characterize each page, and then to identify the most frequent words using a clustering model. This simple example can be easily extended to a more comprehensive document-clustering problem where we would use the most common words occurring in a document as flags to group multiple documents. The predictive objective of this exercise is to then use the process to identify any random webpage and determine if the page pertains to one of the two categories which the model has been trained to identify.

The site (http://www.detroitperforms.org) we are looking into is hosted by a public television station and is meant to be used as a platform for reaching out to members of the local community who are interested in the arts and culture. The site serves as a medium for the station to not only engage with community members, but also to eventually aid in targeted marketing campaigns meant to attract donors to public broadcasting. The site has pages for several related categories: Music, Dance, Theatre, Film, and so on. Each of these pages contains articles and events related to that category. Our goal is to characterize each page on the site and identify the top keywords that appear on each page. To that end, we will crawl each category page, extract the content, and convert the information into a structured document vector consisting of keywords. Finally, we will run a k-medoids clustering process to sort the keywords and rank them. Medoid clustering is similar to the k-means clustering described in Chapter 7. A "medoid" is the most centrally located object in a cluster (Park, 2009). K-medoids are less susceptible to noise and outliers when compared to k-means. This is because

k-medoids tries to minimize dissimilarities rather than Euclidean distances, which is what k-means does.

Before we begin webpage clustering in RapidMiner, you need to make sure that the web mining and text mining extensions are installed. *(This is easily done by going to Help → Updates and Extensions on the main menu bar.)* RapidMiner provides three different ways to crawl and get content from websites. The *Crawl Web* operator will allow setting up of simple crawling rules and based on these rules will store the crawled pages in a directory for further processing. The *Get Page* operator retrieves a single page and stores the content as an example set. The *Get Pages* operator works similarly, but can access multiple pages identified by their URLs contained in an input file. We will use the *Get Pages* operator in this example. Both of the *Get Page(s)* operators allow the choosing of either the GET or POST HTTP request methods for retrieving content.[2]

Step 1: Gather Unstructured Data

The first step in this process is to create an input text file containing a list of URLs to be scanned by the *Get Pages* operator. This is specified in the *Read CSV* (renamed in the process shown in Figure 9.6a to *Read URL List*) operator, which initiates the whole process. The text file consists of three lines: a header line that is needed for the link attribute parameter for Get Pages and two lines containing the two URLs that we are going to crawl, as shown in Figure 9.5 below.[3] The first URL is the "Dance" category page and the second one is the "Film" category page on the website. Save the text file as "pages.txt" as shown in the figure.

The output from the *Get Pages* operator consists of an example set that will contain two main attributes: the URL and extracted HTML content. Additionally it also adds some metadata attributes that are not needed in this example,

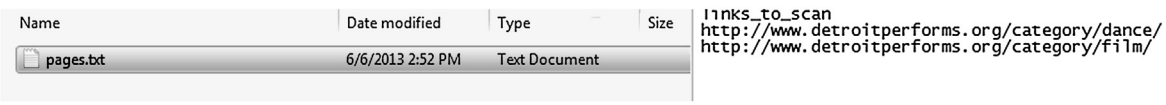

FIGURE 9.5
Creating a URL read list.

[2]For more information on the differences between the two methods, and when to use which type of request, refer to the tutorials on www.w3schools.com.
[3]Be aware that websites may frequently change their structure or content or be taken down altogether. The results shown here for this example were obtained when the website listed was crawled at the time of writing. Your results may differ depending upon when the process is executed.

such as content length (characters), date, and so on. We can filter out these extra attributes using the *Select Attributes* operator.

Step 2: Data Preparation

Next, connect the output from this to a *Process Documents from Data* operator. This is a nested operator, which means this operator contains an inner subprocess where all the preprocessing takes place. The first step in this preprocessing is removing all the HTML tags and only preserving the actual content. This is enabled by the *Extract Content* operator. Put this operator inside the *Process Documents from Data* operator and connect the different operators as shown in Figures 9.6a and 9.6b. Refer to Table 9.3 from earlier to see which operators to use. The inset shows the operators inside the nested *Process Documents from Data* operator. In this case, we will need to use the word occurrences for the clustering. So you need to select Term Occurrences for the vector creation parameter option when configuring the *Process Documents from Data* operator.

Step 3: Apply Clustering (Descriptive Analytics) Technique

The output from the *Process Documents from Data* operator consists of (1) a word list and (2) a document vector or TDM. The word list is not needed for clustering, however the document vector is. Recall that the difference between the two is that in the document vector, each word is considered an attribute and each row or example is a separate document (in this case the web pages crawled). The values in the cells of the document vector can of course be word occurrences, word frequencies, or TF–IDF scores, but as noted in step 2, in this case the cells will have word occurrences. The output from the *Process Documents from Data* operator is filtered further to remove attributes that are less than 5 (that is all words that occur less than five times in *both* documents). Notice that RapidMiner will only remove those attributes (words) which occur less than five times in *both* documents—for example the word "dance" appears only two times in the Film category, but is the most common word in the Dance; it is not and should not be removed! Finally this cleaned output is fed into a k-medoids clustering operator, which is configured as shown in Figure 9.6c.

Upon running the process, RapidMiner will crawl the two URLs listed and execute the different operations to finally generate two clusters. To view these clustering outputs, you can select either the Centroid Table or Centroid Plot views in the Cluster Model (Clustering) results tab, which will clearly show the top keywords from each of the two pages crawled. In Figure 9.7, we see the top few keywords that characterize each cluster. One can then use this model to identify if the content of any random page would belong to either one of the categories.

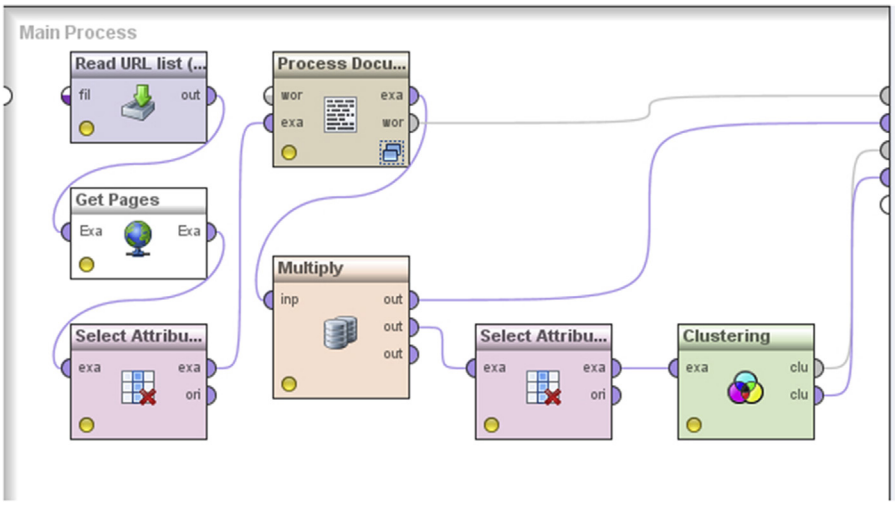

FIGURE 9.6a
Overall process of creating keyword clustering from websites.

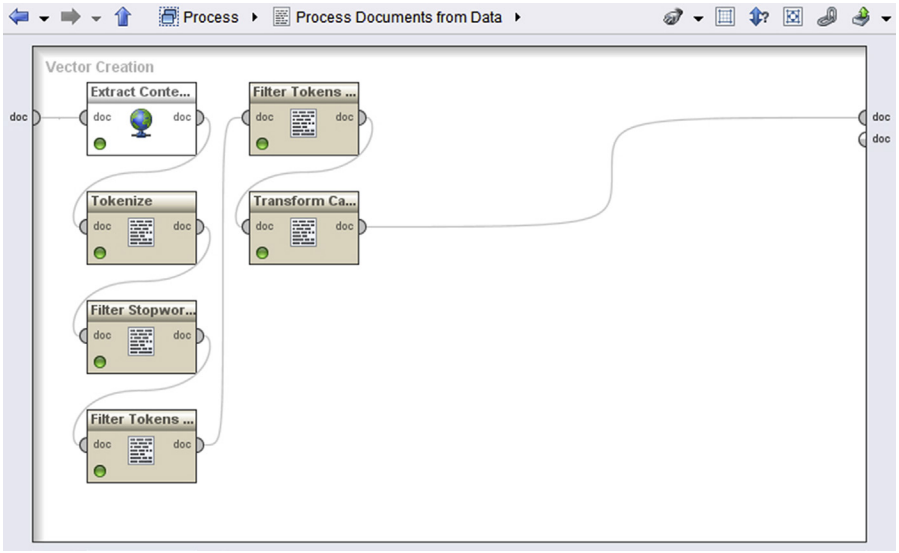

FIGURE 9.6b
Configuring the nested preprocessing operator: Process Documents from Data.

9.2.2 Case Study 2: Predicting the gender of blog authors

The objective of this case study is to attempt to predicting the gender of blog authors based on the content of the blog.

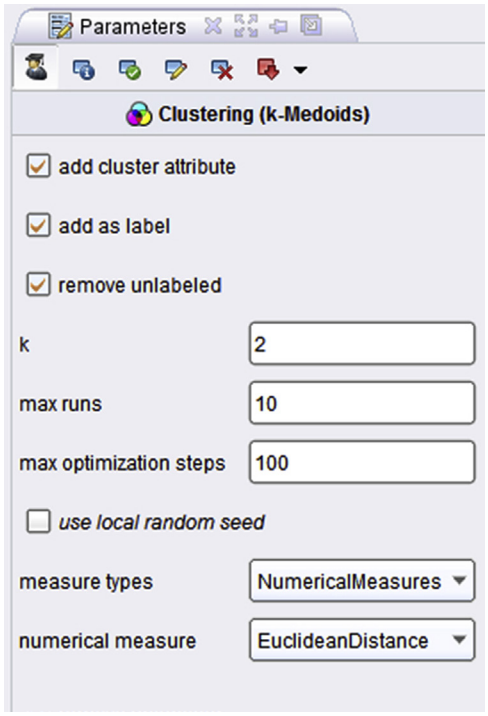

FIGURE 9.6c
Configuring the k-medoids operator.

Step 1: Gather Unstructured Data

The data set for this case study[4] consists of more than 3,000 individual blog entries (articles) by men and women from around the world (Mukherjee, 2010). The data is organized into a single spreadsheet consisting of 3,227 rows and two columns as shown in the sample in Table 9.4. The first column is the actual blog content and the second column is the author's gender, which has been labeled.

For the purpose of this case study, we will split the raw data into two halves: the first 50% of the data is treated as training data with known labels and the remaining 50% is set aside to verify the performance of the training algorithm.

[4]A compressed version of this data can be downloaded from the *Opinion Mining, Sentiment Analysis and Opinion Spam Detection* website (http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html). The data set is called *Blog Author Gender Classification dataset associated with the paper (Mukherjee and Liu, EMNLP-2010)*. This site is maintained by Prof. Bing Liu of the University of Illinois at Chicago. This site contains a lot of relevant information related to text mining and sentiment analysis, in addition to several other useful data sets.

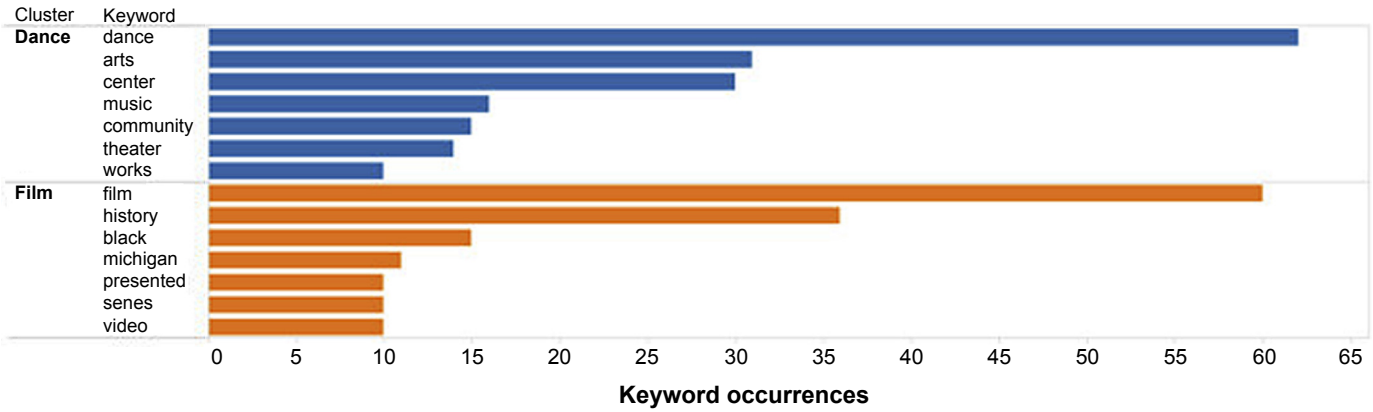

FIGURE 9.7

Results of the website keyword clustering process.

Table 9.4 Raw Data for the Blog Classification Study

BLOG	GENDER
This game was a blast. You (as Drake) start the game waking up in a train that is dangling over the side of a cliff. You have to climb up the train car, which is slowly teetering off the edge of the cliff, ready to plummet miles down into a snowy abyss. From the snowy beginning there are flashbacks to what led Drake to this predicament. The story unfolds in a very cinematic manner, and the scenes in between levels, while a bit clichéd by Hollywood standards, are still just as good if not better than your average brainless Mel Gibson or Bruce Willis action movie. In fact, the cheese is part of the fun and I would venture to say it's intentional.	M
My mother was a contrarian, she was. For instance, she always wore orange on St. Patrick's Day, something that I of course did not understand at the time, nor, come to think of it do I understand today. Protestants wear orange in Ireland, not here, but I'm pretty sure my mother had nothing against the Catholics, so why did she do it? Maybe it had to do with the myth about Patrick driving the snakes, a.k.a. pagans, out of Ireland. Or maybe it was something political. I have no idea and since my mother is long gone from this earth, I guess I'll never know.	F
LaLicious Sugar Soufflé body scrub has a devoted following and I now understand why. I received a sample of this body scrub in Tahitian Flower and after one shower with this tub of sugary goodness, I was hooked. The lush scent is deliciously intoxicating and it ended up inspiring compliments and extended sniffing from both loved ones and strangers alike. Furthermore, this scrub packs one heck of a punch when it comes to pampering dry skin. In fact, LaLicious promises that this body scrub is so rich that it will eliminate the need for applying your post-shower lotion. This claim is true — if you follow the directions.	F
Stopped by the post office this morning to pick up a package on my way to the lab. I thought it would be as good a time as any to clean up my desk and at the very least make it appear that I am more organized than I really am (seriously, it's a mess). It's pretty nice here on the weekends, it's quiet, there's less worry of disturbing undergrad classes if I do any experiments in the daytime. Anyway, it turns out the t-shirt I ordered from Concrete Rocket arrived! Here's how the design looks:	M
See here's the thing: Men have their neat little boxes through which they compartmentalize their lives. Relationship over? Oh, I'll just close that box. It's not that easy for women. Our relationships are not just a section of our lives–they run through the entire fabric, a hot pink thread which adds to the mosaic composing who we are. Take out a relationship and you grab that thread and pull. Have you ever pulled a thread on a knit sweater? That's what it's like. The whole garment gets scrunched and disfigured just because that one piece was removed. And then you have to pull it back apart, smooth it out, fill in the gaps. See here's the thing: men have their neat little boxes through which they compartmentalize their lives. Relationship over? Oh, I'll just close that box. It's not that easy for women.	F
I had pretty bad vision since I was in 3rd grade. About 5 years ago, after watching many people around me get Lasik, I decided to take the plunge. Because this was an elective surgery that in rare cases, could cause damage to my eyes, I decided to find the absolute best doctor for the job. I chose Dr Coleman Kraff and he was amazing to watch. Very confident, poised, fast, and focused.... well like a laser.	M

Then there's my work life. I had an incredible review from my boss 2 weeks ago, and have been slowly adding more and more to my occupational plate as opportunities have been presenting themselves. I'm now going to be the head coach for 7th/8th grade girls' volleyball in the fall. I'm working at the Harlem School of the Arts this summer, just two mornings a week, but they've tentatively asked me if I'd be interested in working their Saturday program during the school year. I have three new piano students for the fall – kids who are my students at school and will now also be my private piano students. I had my first lesson today with one of them, and it was incredible.	F
We had problems in the past with contacts due to dryness and the hassle of always taking them out when I sleep. With Acuvue Oasys soft contact lenses I've had more comfort then ever. The dryness is gone. I can wear them longer. I can even sleep in them and not have to take them out every night. I just put in drops when I wake up and I'm good for the day. It's an ideal product for someone who is always on the go and has no energy to do anything but go to bed in the evening. At an affordable price you can't beat them, especially when you use up a new box less often.	F
This camcorder is really good for the price and it has an easy button for people who are not good with camcoders. There is also a touch screen in the camcorder ang you can turn the screen so you can see yourself. It also has a built in lens cover and you don't have to worry about losing it.	M
At the time Debbie and I moved to Rapid City our family started to grow by one boy and four girls. I'd like to introduce You to my "little kids". On the top row starting left to right, my son Brian and my daughter Leslie. On the bottom row; left-to-right, my youngest daughter Shelley, my eldest daughter Christy and my next eldest daughter Suzanne. They are all adults and they all live in Rapid City.	M
Ideally, the process for buying a laptop would involve a single question: "Which laptop should I buy?" The answer would then spring forth from the heavens or your favorite technology Web site. Unfortunately, finding the right laptop for your budget and needs involves answering not one but a series of questions. Fear not, laptop buyer, we know which questions to ask, the answers to those questions, along with the current market trends and where laptops are headed.	M
Ok, so how stinkin' cute IS this bunny?? I saw an adorable Easter sign at Hobby Lobby a few weeks ago and thought it would make a great card. I can't find the adapter that allows me to download photos from my camera phone, so I can't show you the inspiration piece. But it was an Easter bunny peeking over the top of an Easter egg.	F
Slept until 11am. Time change seems to have no effect. In fact filled my tires with air and went for a bike ride around the complex like 5 times. Legs were hurting so stopped and came inside. Time change no effect as of yet but being cautious. Feeling pretty up and positive. Need to get gas for car and then run a few errands. Laundry is going to wait again. Goal for next week is to complete all of it. It is at least in a semi organized piles and ready to be washed. Need to find a laundry shop to complete. Also need to register for IML volunteer and start working on getting taxes filed and the $5,000.00 medical bill from last hospitalization. That is approximate including Doctors which I have yet to receive. Funny they are charging me over $3,000 for the CPAP. that is the price of a new machine. I plan on fighting that charge to see what happens. Will be using health advocate to see what, if anything can be done. Last night did not sleep right away so created contracts for my support system alerting them if what they should be on the look out for and what actions to take if they see me doing those actions.	M

Continued

Table 9.4 Raw Data for the Blog Classification Study—Cont'd

BLOG	GENDER
We celebrated St. Patty's Day with Ethan, Carys, and Molly and..... a dinner made 'o green! We had green pancakes with shamrock-shaped whip cream on top, green vanilla yogurt (green food coloring and I were special friends today) topped with green sprinkles, and honey dew. For dessert, we had a magically delicious treat I referred to as Pots o' Gold - halved orange peels filled with "gold" (yellow jello squares) and garnished with gold coins (chocolate Easter candy)! As for beverages, we had Sprite in clear cups with green Sprite ice cubes to color our drinks. It really was so much fun to get into the spirit of St. Patty's Day.	F
I'm so excited about the gorgeous weather we've been having lately. I feel like spring has crept up early, quickly and unexpectedly this year. But maybe that's just because I was prepared for it to be especially late in coming up here in Sudbury. I'm not getting my hopes up though that it's here to stay, because we'll likely have some more winter before it's gone for good.	F
I own a 9 1/2 year old German Shepherd mix. We believe his father was a black lab with chow mix. His mother was a full-blooded German Shepherd. Fred is a great dog. He weighs 110 lbs. So you can't ever pick him up and his meds cost a fortune, but he's worth every time. I watch a lot of Court TV and I feel much more secure having Fred in the house with me!He's settled down a lot over the years. He used to be much more high strung and jumped on people, etc. Now he just barks a lot. He's definitely the alpha dog of our family.	F
Yesterday we had dense fog, heavy rain last night and fog again this morning. It is supposed to transition (whatever happened to just saying "change"?) to rain beginning around ten this morning. And yes, today is the day we load carts onto the Bookmobile, unload them, reload them, unload them, reload them – all day long. This will be my first experience dealing with that in the rain. I understand that we have some flimsy tarps to put over the carts, but the word is that they are not all that helpful. At least it is supposed to get up to about 53 (11.6 C).	M
Apparently if the rain we are getting this weekend were snow, it would be about three feet deep. It is also pretty windy. Therefore the streets are awash with dead umbrellas rolling like tumbleweed across streets and frightening the dogs. The corpses are flung every which way including upside down. A few are neatly disposed of in garbage receptacles. Others look positively dangerous.	F

While developing models involving large amounts of data, which is common with unstructured text analysis, it is a good practice to divide the process into several distinct processes and store the intermediate data, models, and results from each process for recall at a later stage. RapidMiner facilitates this by providing special operators called *Store* and *Retrieve*. The *Store* operator stores an IO Object in the data repository and *Retrieve* reads an object from the data repository. The use of these operators is introduced in the following sections.

Step 2: Data Preparation

Once the data is downloaded and uncompressed, it yields a single MS Excel file, which can then be imported into the RapidMiner database using the *Read Excel* operator. The raw data consists of 290 examples that do not have a label and one example that has no blog content but has a label! This needs to be cleaned up. It is easier to delete this entry in the raw data—simply delete the row (#1523) in the spreadsheet that contains this missing entry and save the file before reading it into RapidMiner. Also make sure that the *Read Excel* operator is configured to recognize the data type in the first column as "text" and not polynominal (default) as shown in Figure 9.8. Connect the output from *Read Excel* to a *Filter Examples* operator, where will then remove the entries with missing labels. (If you are so inclined, you may want to store these entries with missing labels for use as testing samples—you can accomplish this with another *Filter Examples*, but by checking *Invert Filter* box and then storing the output. In this case however, we will simply discard examples with missing labels.) We can now separate the cleaned data with a 50/50 split using a *Split Data* operator. Save the latter 50% testing data (1,468 samples) to a new file with a *Write Excel* operator and pass the remaining 50% training portion to a *Process Documents from Data* operator.

This, as we now know, is a nested operator where all the preprocessing happens. Recall that this is where the conversion of unstructured data into a structured format will take place. Connect the different operators within as shown in Figure 9.9a. The only point to note here is that you will need a *Filter Stopword (Dictionary)* operator to remove any "nbsp" (" " is used to represent a nonbreaking space) terms that may have slipped into the content. Create a simple text file with this keyword inside it and let RapidMiner know that this dictionary exists by properly configuring the operator. To configure the *Process Documents from Data* operator, use the options as shown in Figure 9.9b.

The output from the process for step 2 consists of the document vector and a word list. While the word list may not be of immediate use in the subsequent steps, it is a good idea to store this along with the very important document vector. The final process is shown in Figure 9.9c.

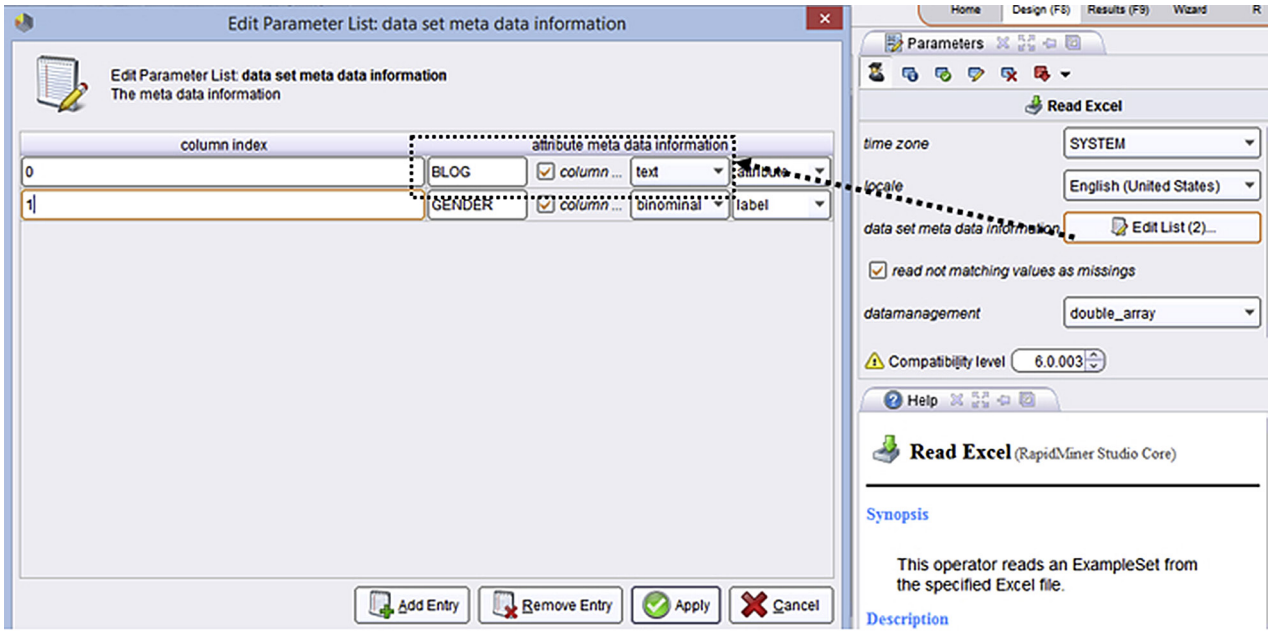

FIGURE 9.8

Properly configuring the Read Excel operator to accept text (not polynomial).

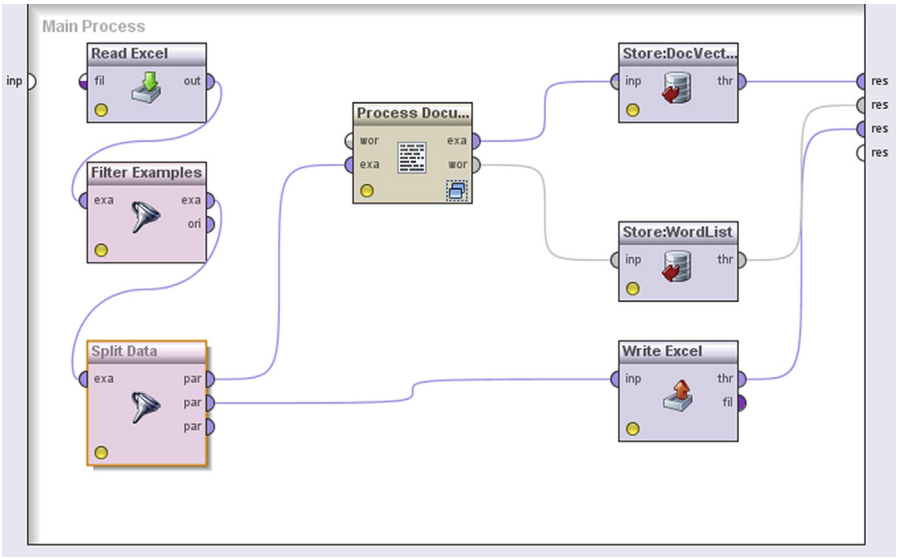

FIGURE 9.9a
Preprocessing text data using the Process Documents from Data operator.

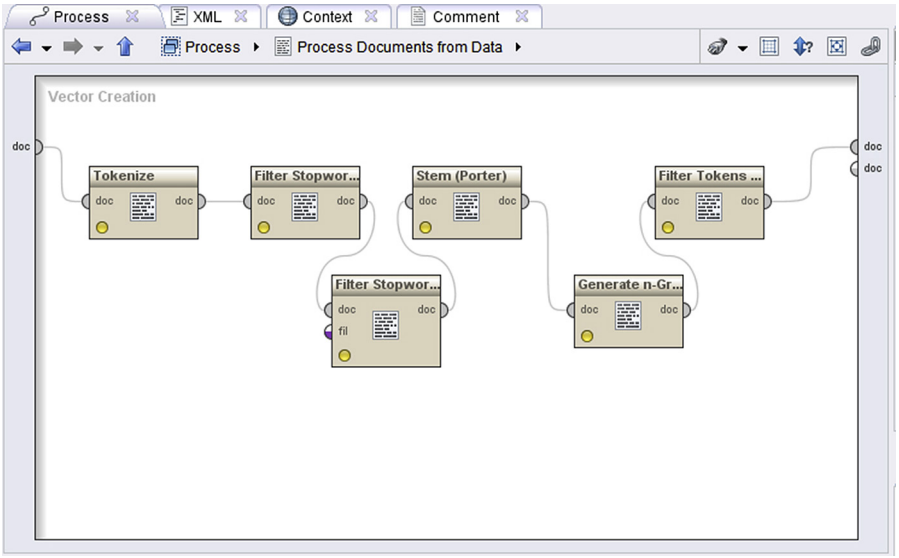

FIGURE 9.9b
Configuring the preprocessing operator.

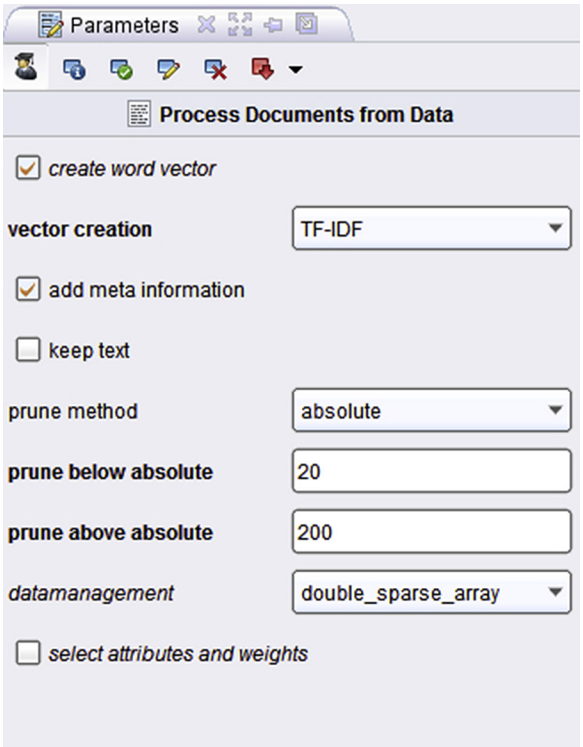

FIGURE 9.9c
Overall process for blog gender classification.

Step 3.1: Identify Key Features

The document vector that is the result of the process in step 2 is a structured table consisting of 2,055 rows—one for every blog entry in the training set—and 2,815 attributes or columns—each token within an article that meets the filtering and stemming criteria defined by operators inside *Process Documents* is converted into an attribute. Training learning algorithms using 2,815 features or variables is clearly an onerous task. The right approach is to further filter these attributes by using feature selection methods.

We will employ two feature selection methods using the *Weight by Information Gain* and *Weight by SVM* operators that are available. *Weight by Information Gain* (more details in Chapter 12 Feature Selection on this operator) will rank a feature or attribute by its relevance to the label attribute (in this case, gender) based on the information gain ratio and assigns weights to them accordingly. *Weight by SVM* will set the coefficients of the SV hyperplane as attribute weights. Once we rank them using these techniques, we can select only a handful of attributes (for example, the top 20) to build our models. Doing so will result in a reasonable reduction in modeling costs.

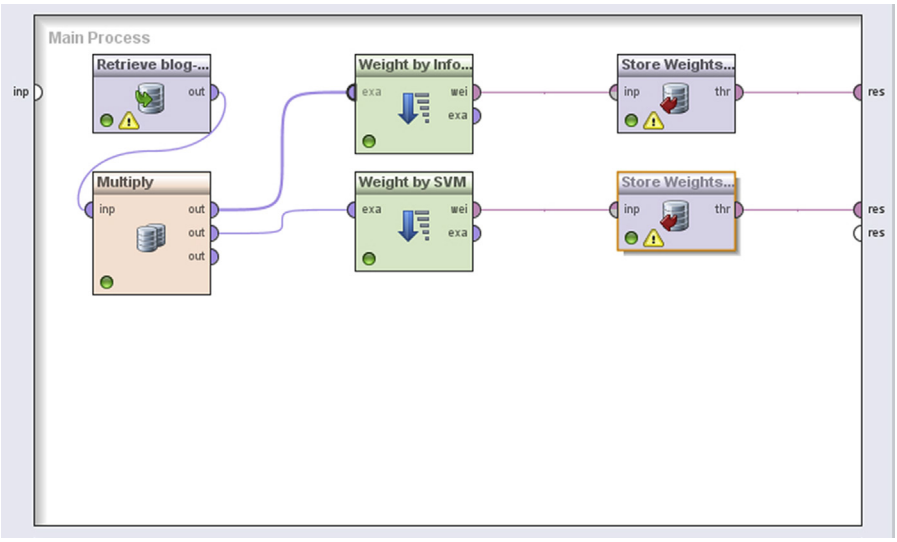

FIGURE 9.10
Using feature selection methods to filter attributes from the TDM.

The results of this intermediate process will generate two weight tables, one corresponding to each feature selection method. We start the process by retrieving the document vector saved in step 2 and then end the process by storing the weight tables for use in step 3.2 (see Figure 9.10).

In the paper by Mukherjee and Liu (Mukherjee, 2010) from which this data set comes, they demonstrate the use of several other feature selection methods, including a novel one developed by the authors that is shown to yield a much higher prediction accuracy than the stock algorithms (such as the ones we demonstrate here).

Step 3.2: Build Predictive Models

Once we have the document vector and attribute weights, we can experiment using several different machine learning algorithms to understand which give the best accuracy. The process illustrated in Figures 9.11a and b will generate the models and store them (along with the corresponding performance results) for later application. This is one of the key strengths of RapidMiner: once we have built up the necessary data for predictive modeling, switching back and forth between various algorithms requires nothing more than dragging and dropping the needed operators and making the connections. As seen in Figure 9.11b, we have five different algorithms nested inside the *X-Validation* operator and can conveniently switch back and forth as needed. Table 9.5 shows that the *LibSVM(linear)* and *W-Logistic* operators (Available through Weka extension for RapidMiner. Go to Help > Updates and Extensions) seem to give the best

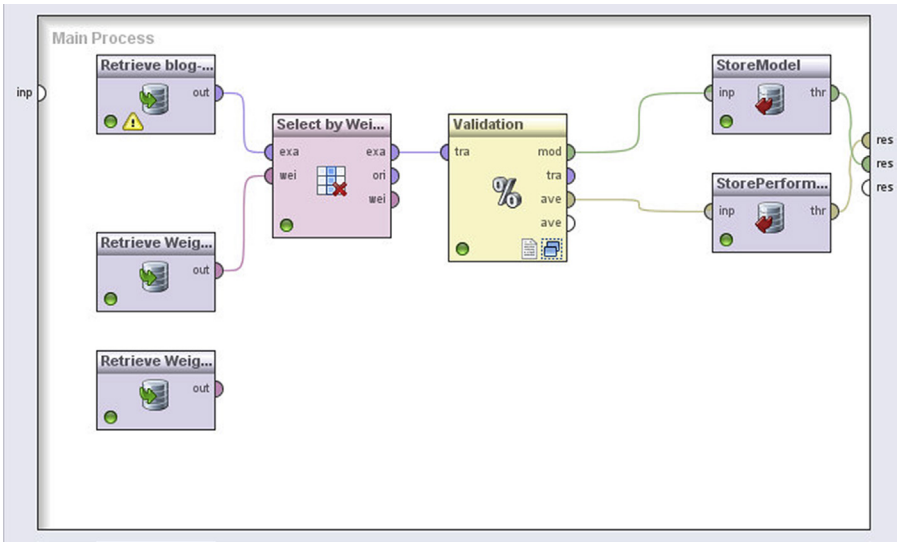

FIGURE 9.11a
Training and testing predictive models for blog gender classification.

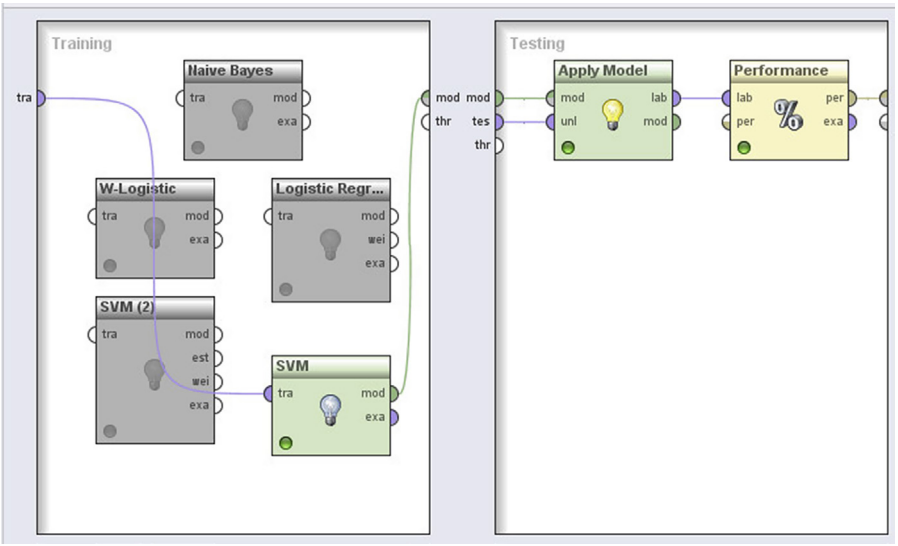

FIGURE 9.11b
Switching between several algorithms.

Table 9.5 Comparing the Performance of Different Training Algorithms for Blog Gender Classification

Algorithm	Class Recall (M)	Class Recall (F)	Accuracy
LibSVM (linear)	87	53	72
W-Logistic	85	58	73
Naïve Bayes	86	55	72
SVM (polynomial)	82	42	63

performance. Keep in mind that these accuracies are still not the highest and are in line with the performances reported by Mukherjee and Liu in their paper for generic algorithms.

To improve upon these, we may need to further optimize the best performers so far by nesting the entire validation process within an optimization operator. This is described in the chapter 13, in the section on optimization.

Step 4.1: Prepare Test Data for Model Application

Going back to the original 50% of the "unseen" data that was saved for testing purposes, we can actually evaluate the real-world performance of the best algorithm in classifying blogs by author gender. However, keep in mind that we cannot use the raw data that we set aside as is (what would happen if you did?). We need to also convert this raw test data into a document vector first. In other words, we need to repeat the step 2 process (without the filtering and split data operators) on the 50% of data that was set aside for testing. The *Process Documents from Data* operator can be simply copied and pasted from the process in step 2. (Alternatively, we could have preprocessed the entire dataset before splitting!) The document vector is stored for use in the next step. This process is illustrated in Figure 9.12.

Step 4.2: Applying the Trained Models to Testing Data

This is where the rubber hits the road! The last step will take any of the saved models created in step 3.2 and the newly created document vector from step 4.1 and apply the model on this test data. The process is shown below in Figures 9.13a and b. One useful operator to add is the *Set Role* operator, which will be used to indicate to RapidMiner the label variable. Doing so will allow us to sort the results from the Apply Model by "Correct Predictions" and "Wrong Predictions" using the View Filter in the Results perspective as shown here.

When you run this process you will find that the LibSVM (linear) model can correctly predict only 828 of the 1,468 examples, which translates to a poor 56% accuracy! The other models fare worse. Clearly the model and the process are in need of optimization and further refinement. Using RapidMiner's

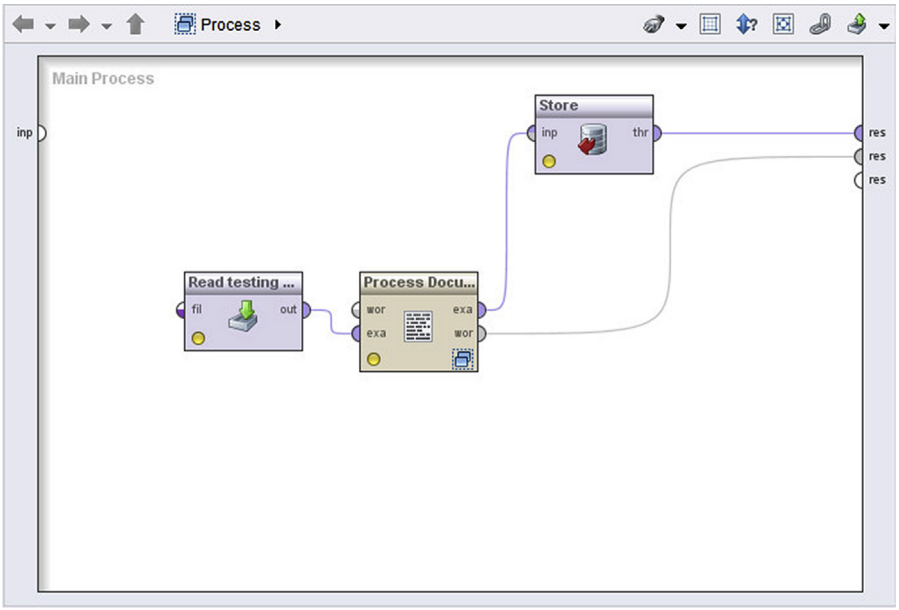

FIGURE 9.12
Preparing the "unseen" data for model deployment.

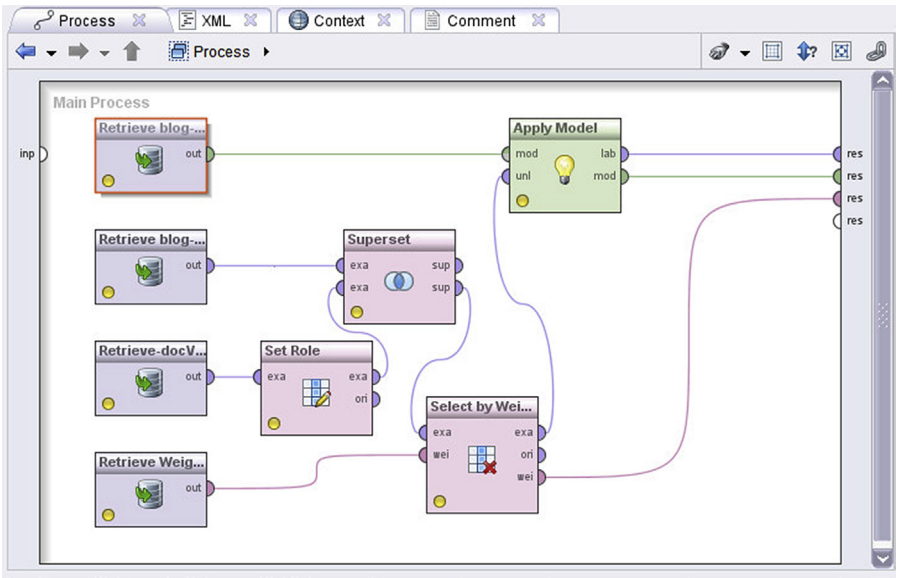

FIGURE 9.13a
Applying the models built in step 3 on the unseen data.

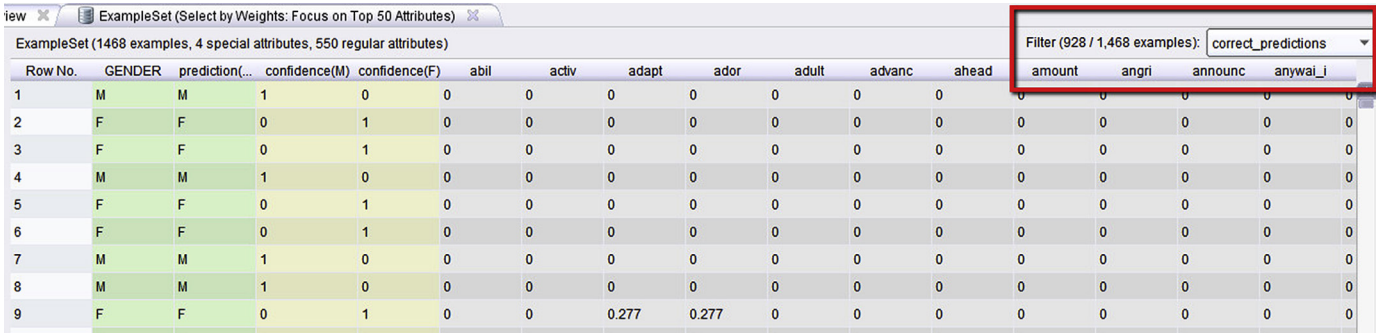

FIGURE 9.13b

The results view.

built-in optimization operators, one can easily improve upon this baseline accuracy. A discussion about how to use the optimization operators in general is provided in Chapter 13 Getting Started with RapidMiner. The truly adventurous can implement the Mukherjee and Liu algorithm for feature selection in RapidMiner based on the instructions given in their paper!

CONCLUSION

Unstructured data, of which text data is a major portion, appears to be doubling in volume every three years (Mayer-Schonberger, 2013). The ability to automatically process and mine information from such digital data will become an important skill in the future. These techniques can be used to classify and predict just as the other techniques throughout the book, except we are now working on text documents and even voice recordings that have been transcribed to text.

In this chapter we explained how unstructured data can be mined using any of the available algorithms presented in this book. The key to being able to apply these techniques is to convert the unstructured data into a semi-structured format. We introduced a high-level three-step process that will enable this. We discussed some key tools for transforming unstructured data, such as tokenization, stemming, n-gramming, and stopword removal. We then discussed how concepts such as TF–IDF will allow us to make the final transformation of a corpus of text to a matrix of numbers, which can be worked upon by the standard machine learning algorithms. Finally, we presented a couple of real-world examples, which will allow you to explore the exciting world of text mining using RapidMiner.

REFERENCES

Cutting, D. K. (1992). Scatter/gather: A cluster-based approach to browsing large document collections. *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 318–329 Copenhagen.

Franz, A. A. (2006, August 3). *All our N-gram are Belong to You*. Retrieved November 1, 2013, from Research Blog, http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html.

Hearst, M. (June 20–26, 1999). *Untangling Text Data MiningProceedings of Association for Computational Linguistics, 37th Annual Meeting 1999*. University of Maryland.

International Monetary Fund (n.d.). Retrieved from http://www.imf.org/external/pubs/ft/weo/2012/02/weodata/index.aspx.

Mayer-Schonberger, V. A. (2013). *Big Data: A Revolution That Will Transform How We Live, Work and Think*. London: John Murray and Co.

McKnight, W. (2005, January 1). *Text Data Mining in Business Intelligence*. Retrieved November 1, 2013, from Information Management, http://www.information-management.com/issues/20050101/1016487-1.html#Login.

Park, H. S., Jun, C. H. (2009). A simple and fast algorithm for K-medoids clustering. *Expert Systems with Applications*. 36(2), 3336–3341.

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

Mukherjee, A. L. (2010). Improving Gender Classification of Blog Authors. *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP-10)* Cambridge, MA.

Siegel, E. (2013). *Predictive Analytics: The Power to Predict Who Will Click, Buy, Lie or Die*. Hoboken, NJ: John Wiley and Sons.

Upbin, B. (2013, February 8). IBM's Watson gets its first piece of business in Healthcare. *Forbes*.