CHAPTER 9

Text Mining

This chapter explores and formalizes how to extract patterns and discover new knowledge by applying many of the techniques learned so far, not on ordered data, but on unstructured natural language. This constitutes the vast 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 are the same techniques applicable to extract patterns and predict outcomes when the input data looks like normal written communication? This might seem baffling at first, but as shall be seen in this chapter, there are ways of presenting text data to the same algorithms that process normal data.

To start out a brief historical introduction to the field of text mining is given to establish some context. In the following section, techniques that can convert common text into a semi-structured format will be described that we, and the algorithms introduced so far, can recognize. Finally, the text mining concepts will be implemented using two case studies: one involving an unsupervised (clustering) model and another involving a supervised support vector machine (SVM) model. The chapter will be closed with some key considerations to keep in mind while implementing text mining.

Unstructured data (including text, audio, images, videos, etc.) is the new frontier of data science. 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 science.

Data Science. DOI: https://doi.org/10.1016/B978-0-12-814761-0.00009-5

Some of the first applications of text mining came about when people were trying to organize documents (Cutting, 1992). Hearst (1999) 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 have been discussed so far.

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 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 will be discussed 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. A couple of these will be explored in this chapter.

People in the data management 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 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, it is an 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 IT WORKS

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

Each of the main processes will now be examined in detail and some necessary terminology and concepts will be introduced. But before these processes are described, a few core ideas essential to text analytics will need to be defined.

9.1.1 Term Frequency—Inverse Document Frequency

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 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, that 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 web pages that contain a large number of instances of the rare keywords.

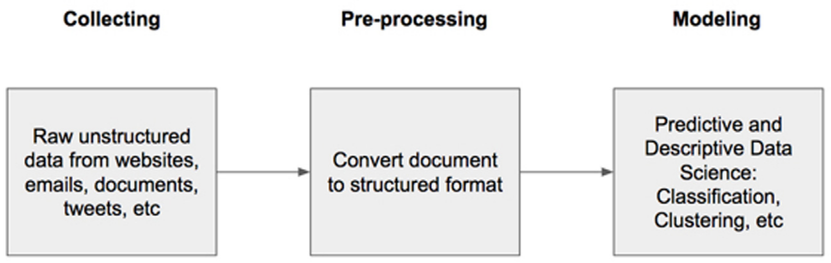

Collecting
Pre-processing
Modeling
Raw unstructured data from websites, emails, documents, tweets, etc
Convert document to structured format
Predictive and Descriptive Data Science: Classification, Clustering, etc

FIGURE 9.1
A high-level process for text mining.

In this 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 7.7 billion results for the word "books," whereas only 584,000 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. 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. 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 also 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 term *TF–IDF*, which stands for term frequency–inverse document frequency.

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

$$\mathrm{TF} = \frac{n_k}{n} \tag{9.1}$$

Considering the mentioned 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:

$$\mathrm{IDF} = \log_2\left(\frac{N}{N_k}\right) \tag{9.2}$$

where N is the number of documents under consideration (in a search engine context, N is the number of all the indexed web pages). For most text mining problems, N is the number of documents that one is trying to mine, and N_k is the number of documents that contain the keyword, k. Again, a word such as "that" would arguably appear in every document and, thus, the ratio (N/N_k) would be close to 1, and the IDF score would be close to zero for. However, a word like "RapidMiner" would possibly appear in a relatively fewer number of documents and so the ratio (N/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:

$$\mathrm{TF} - \mathrm{IDF} = \frac{n_k}{n} \times \log_2\left(\frac{N}{N_k}\right) \tag{9.3}$$

Going back to the mentioned example, when the high TF for "that" is multiplied by its corresponding low IDF, a low (or zero) TF—IDF will be reached, whereas when the low TF for "RapidMiner" is multiplied by its corresponding fairly high IDF, a relatively higher TF—IDF would be obtained.

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 science techniques that have been discussed so far in this book. In the following sections additional concepts that are commonly employed in text mining will be described.

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." Suppose the objective is to perform a comparison between them, or a similarity mapping. For this purpose, each sentence is one unit of text that needs to be analyzed.

These two sentences could be embedded in an email message, in two separate web pages, in two different text files, or else they 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 the 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

Some form of structure can be imposed 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 science parlance and each document is an example. One, therefore, has 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

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

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 a third statement is added, "RapidMiner is offered as an open source software program." This new document will increase the number of rows of the 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 more new statements are added that have little in common, one would end up with a very sparse matrix.

Note that one 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. TF can be obtained 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, one could have also chosen to use the TF–IDF scores for each term to create the document vector. This is also shown in Fig. 9.2.

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

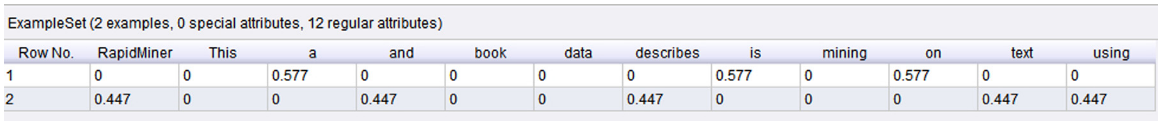
In addition to filtering standard stop words, some specific terms might also need to be filtered out. For example, in analyzing text documents that pertain to the automotive industry, one may want to filter away terms that are common to this industry such as "car," "automobile," "vehicle," and so on. This

[1] RapidMiner does a double normalization while calculating TF scores. For example, in case of Document 1, the TF score for the term "data" would be $(0.1498)/\sqrt{(0.1498^2 + 0.1498^2 + \ldots + 0.1498^2)} = 0.1498/\sqrt{7*(0.1498^2)} = 0.3779$. Similarly for all other terms, double normalization makes it easier to apply algorithms such as SVM. This change in TF calculation is reflected in the TF-IDF scores.

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

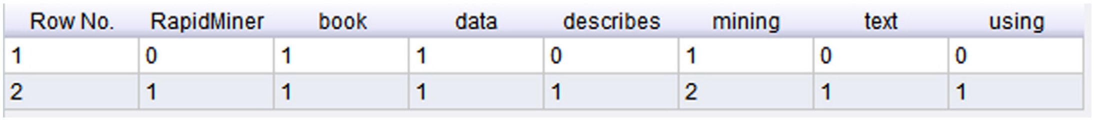
TDM, *Term document matrix.*



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. *TF—IDF*, Term Frequency—Inverse Document Frequency; *TDM*, term document matrix.

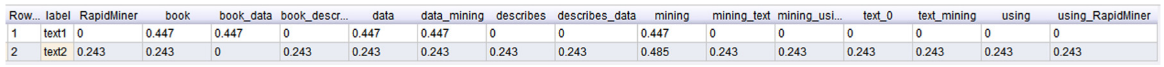


Row No.	RapidMiner	book	data	describes	mining	text	using
1	0	1	1	0	1	0	0
2	1	1	1	1	2	1	1

FIGURE 9.3
Stop word filtering reduces the size of the TDM significantly.

is generally achieved by creating a separate dictionary where these context-specific terms can be defined and then *term filtering* can be applied to remove them from the 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 on the field or subject which is being analyzed—this is especially important in areas with specific jargon, e.g., in clinical settings.)

Words such as "recognized," "recognizable," or "recognition" may be encountered 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, the conversion of unstructured text to structured data can be simplified because now only the occurrence of the root terms has to be taken into account. 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 "anomalies" with "anomaly." 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 extremely 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 meanings. There are other stemmers available; which one is chosen is usually guided by ones experience in various domains. Stemming is usually the next process step following term filtering. (A word of caution: stemming is



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. *TF—IDF*, Term Frequency—Inverse Document Frequency.

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	Stop word 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)

completely dependent on the human language being processed as well as the period of the language being processed. 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 other different uses. 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 the 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 on the size of the documents and the corpus.

Fig. 9.4 shows a TF-based document vector for bigrams (*n* = 2) from the examples and as can be seen, 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 (up-casing and down-casing), or sometimes even performing a simple spell check beforehand. Data quality in text mining is just as important as in other areas.

9.2 HOW TO IMPLEMENT

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

9.2.1 Implementation 1: Keyword Clustering

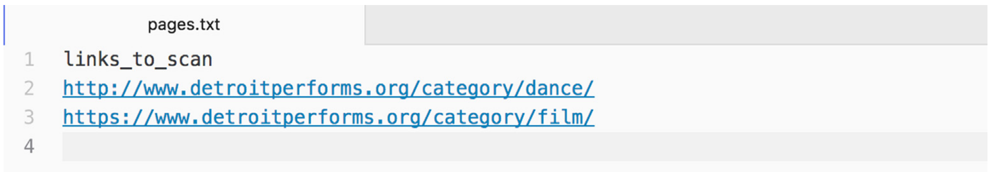
In this first example, some of the web mining features of RapidMiner will be introduced and then a clustering model will be created with keywords data mined from a website. The objective of this case study is to scan several pages from a given website and identify the most frequent words within these 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 the most common words occurring in a document would be used 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) that is being looked 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 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, Theater, Film, and so on. Each of these pages contains articles and events related to that category. The goal here is to characterize each page on the site and identify the top keywords that appear on each page. To that end, each category page will be crawled, the content extracted, and the information converted into a structured document vector consisting of keywords. Finally, a *k*-medoids clustering process will be run to sort the keywords and rank them. Medoid clustering is similar to the *k*-means clustering described in Chapter 7, Clustering. A medoid is the most centrally located object in a cluster (Park & Jun, 2009). *k*-Medoids are less susceptible to noise and outliers when compared to *k*-means. This is because *k*-medoids tries to minimize dis-similarities rather than Euclidean distances, which is what *k*-means does.

Before beginning with web page clustering in RapidMiner, 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. The *Get Pages* operator will be used 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 Fig. 9.6 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 are going to be crawled, as shown in Fig. 9.5.[3]



```
links_to_scan
http://www.detroitperforms.org/category/dance/
https://www.detroitperforms.org/category/film/

```

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.

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, such as content length (characters), date, and so on. These extra attributes can be filtered out 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 Figs. 9.6 and 9.7. 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, the word occurrences will need to be used for the clustering. So, select Term Occurrences for the vector creation parameter option when configuring the *Process Documents from Data* operator.

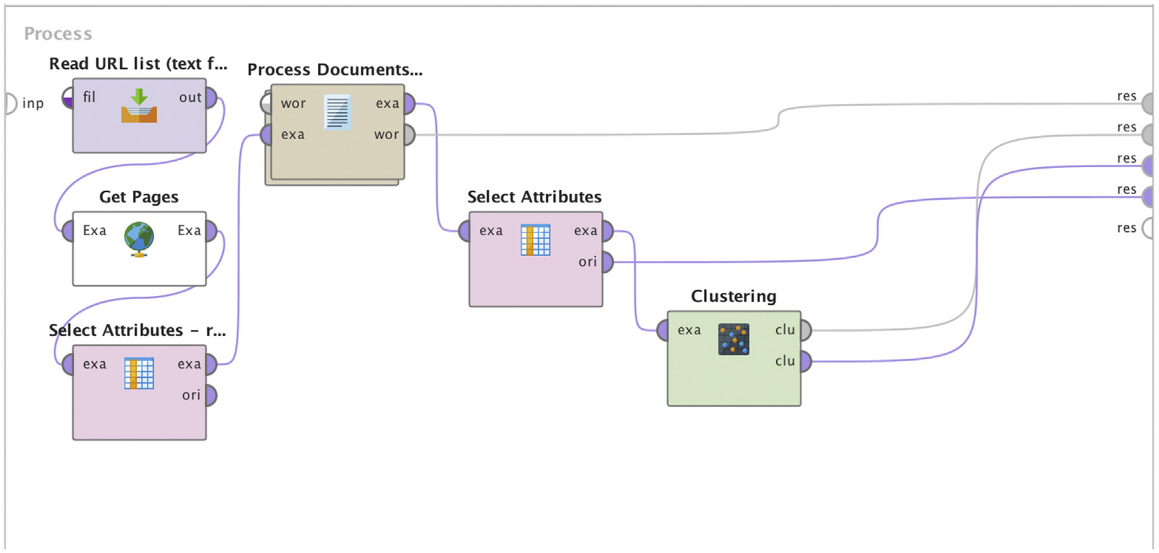


FIGURE 9.6
Overall process of creating keyword clustering from websites.

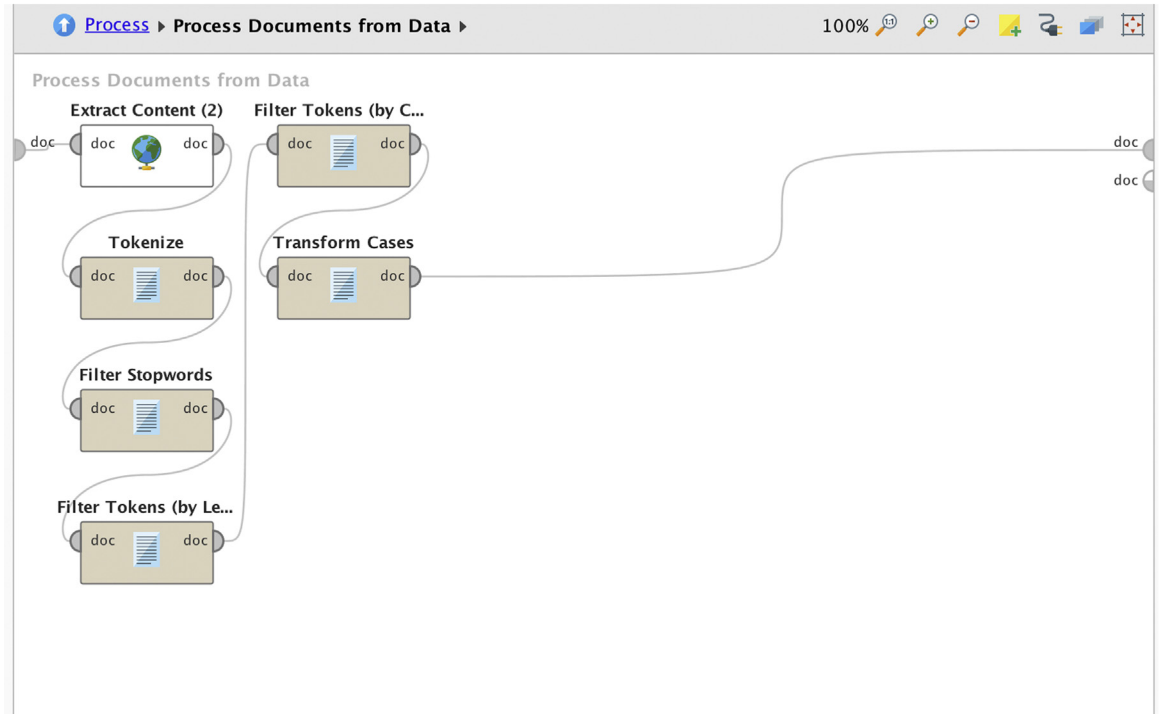
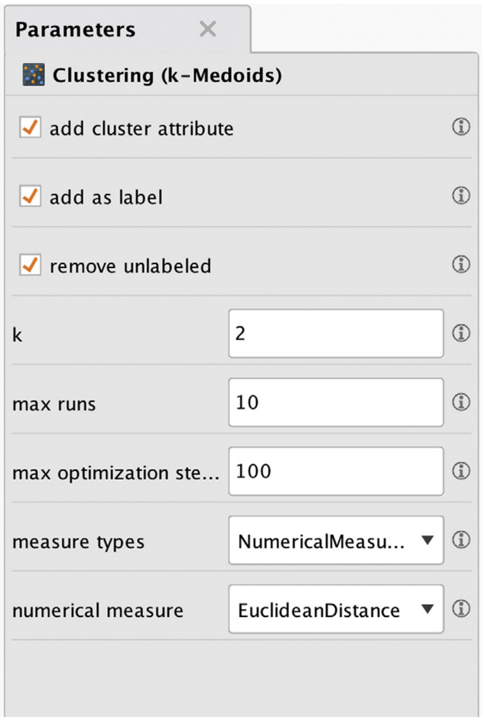


FIGURE 9.7
Configuring the nested preprocessing operator: process documents from data.

Step 3: Apply Clustering

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 category; 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 Fig. 9.8.


Parameters

Clustering (k-Medoids)

✓ add cluster attribute

✓ add as label

✓ remove unlabeled

k	2
max runs	10
max optimization ste...	100
measure types	NumericalMeasu...
numerical measure	EuclideanDistance

FIGURE 9.8
Configuring the *k*-medoids operator.

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, 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 Fig. 9.9, 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.

9.2.2 Implementation 2: Predicting the Gender of Blog Authors

The objective of this case study is to attempt to predict the gender of blog authors based[4] on the content of the blog (should it be predicted? why? can it be predicted?).

[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, 2010). This site contains a lot of relevant information related to text mining and sentiment analysis, in addition to several other useful data sets.

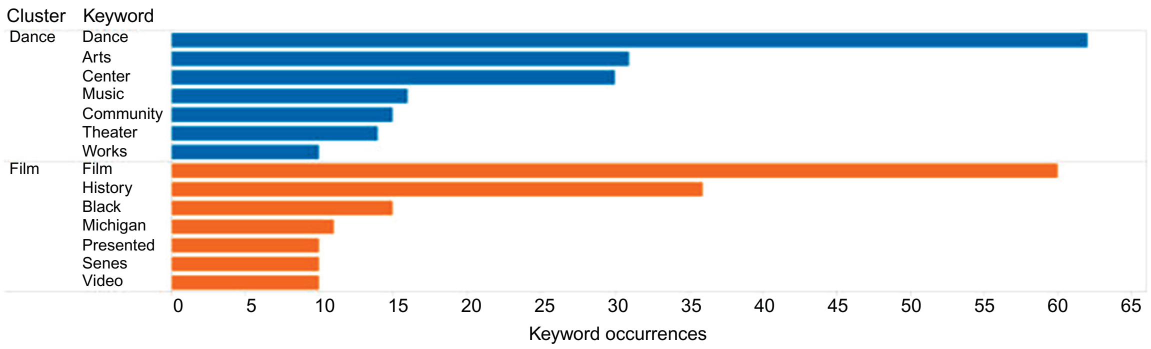


FIGURE 9.9
Results of the website keyword clustering process.

Step 1: Gather Unstructured Data

The data set for this case study consists of more than 3000 individual blog entries (articles) by men and women from around the world (Mukherjee, 2010). The data[4] is organized into a single spreadsheet consisting of 3227 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, the raw data will be split 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.

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 input-ouput (IO) Object in the data repository and *Retrieve* reads an object from the data repository. The use of these operators is introduced in the coming sections.

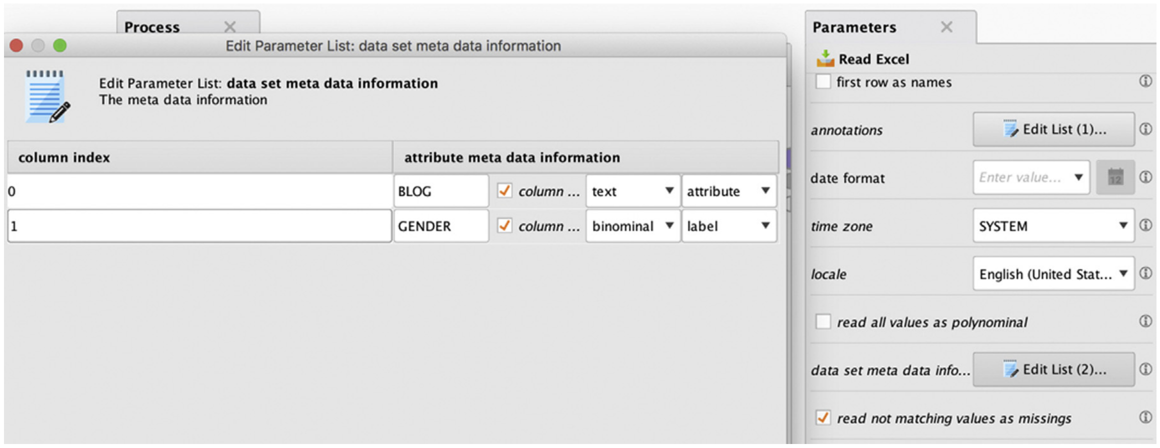
Step 2: Data Preparation

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

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	M
Anyway, it turns out the T-shirt I ordered from Concrete Rocket arrived! Here's how the design looks	
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

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 Fig. 9.10. Connect the output from *Read Excel* to a *Filter Examples* operator, where the entries with missing labels will then be removed. (If inclined, store these entries with missing labels for use as testing samples—this can be accomplished with another *Filter Examples,* but by checking *Invert Filter* box and then storing the output. In this case, however, examples with missing labels will simply be discarded.) The cleaned data can now be separated with a 50/50 split using a *Split Data* operator. Save the latter 50%



Edit Parameter List: data set meta data information

Edit Parameter List: **data set meta data information**
The meta data information

column index	attribute meta data information			
0	BLOG	✓ column ...	text	attribute
1	GENDER	✓ column ...	binominal	label

Read Excel

☐ first row as names	
annotations	Edit List (1)...
date format	Enter value...
time zone	SYSTEM
locale	English (United Stat...
☐ read all values as polynominal	
data set meta data info...	Edit List (2)...
✓ read not matching values as missings	

FIGURE 9.10
Properly configuring the Read Excel operator to accept text (not polynomial).

testing data (1468 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 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 Fig. 9.13. The only point to note here is that a *Filter Stop word (Dictionary)* operator will be needed to remove any "nbsp" ("&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 Fig. 9.11.

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 Fig. 9.12.

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 2055 rows—one for every blog entry in the training set—and 2815 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 2815 features or variables is clearly an onerous task. The right approach is to further filter these attributes by using feature selection methods.

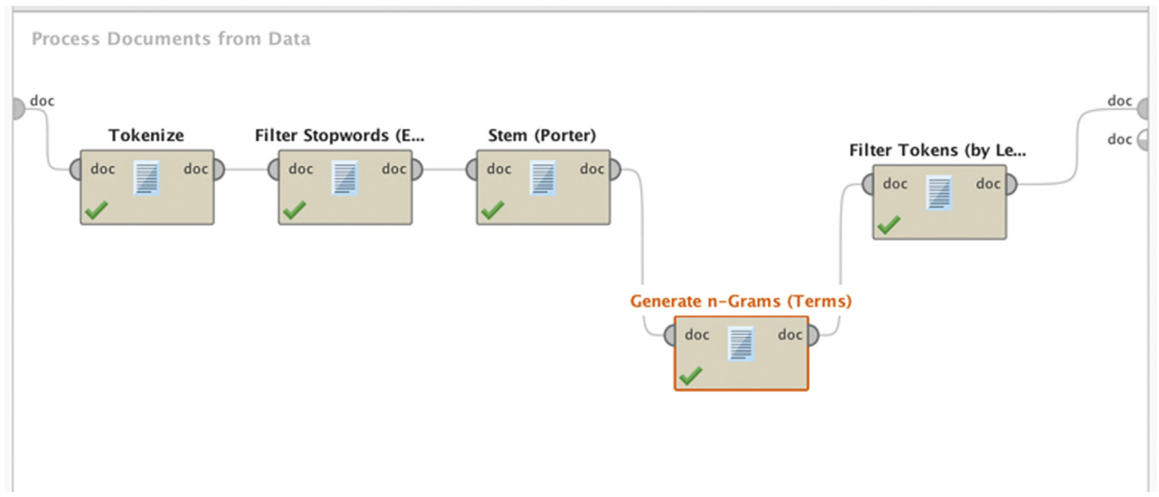


FIGURE 9.11
Configuring the preprocessing operator.

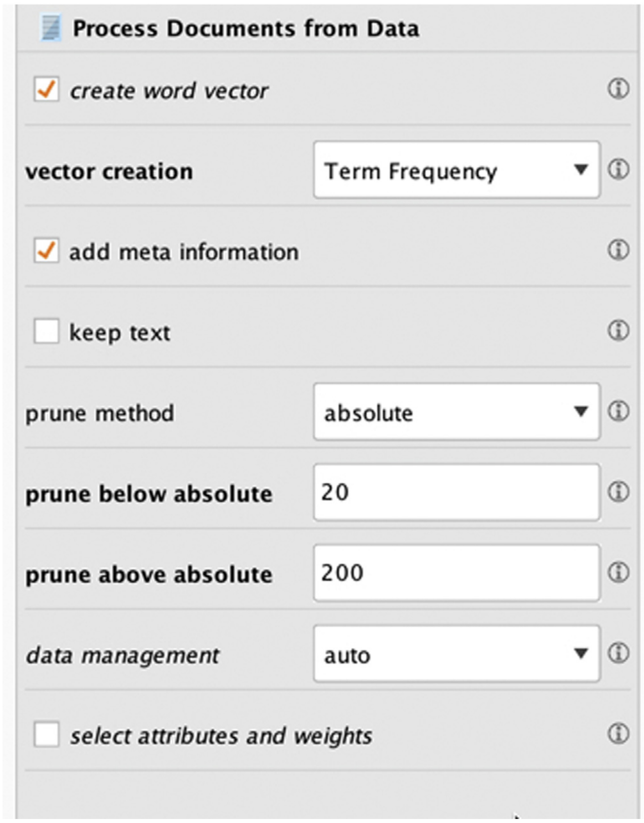
Two feature selection methods will be employed using the *Weight by Information Gain* and *Weight by SVM* operators that are available. *Weight by Information Gain* (more details in Chapter 14: Feature Selection, on this operator) will rank a feature or attribute by its relevance to the label attribute (in this case, gender) based on the information gain ratio and assign weights to them accordingly. *Weight by SVM* will set the coefficients of the SVM hyperplane as attribute weights. Once they are ranked using these techniques, only a handful of attributes can be selected (e.g., the top 20) to build the models. Doing so will result in a reasonable reduction in modeling costs.

The results of this intermediate process will generate two weight tables, one corresponding to each feature selection method. The process is started by retrieving the document vector saved in step 2 and then the process is ended by storing the weight tables for use in step 3.2 (see Fig. 9.14).

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 demonstrated here).

Step 3.2: Build Models

With the document vector and attribute weights now ready, one can experiment using several different machine learning algorithms to understand



Process Documents from Data

✓ create word vector	
vector creation	Term Frequency
✓ add meta information	
☐ keep text	
prune method	absolute
prune below absolute	20
prune above absolute	200
data management	auto
☐ select attributes and weights	

FIGURE 9.12
Overall process for blog gender classification.

which gives the best accuracy. The process illustrated in Figs. 9.15 and 9.16 will generate the models and store them (along with the corresponding performance results) for later application. This is a key strength of RapidMiner: once one has 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 Fig. 9.16, there are four different algorithms nested inside the *X-Validation* operator and can conveniently be switched back and forth as needed. Table 9.5 shows that the *LibSVM (linear)* and *W-Logistic* operators (Available through Weka extension for RapidMiner) seem to give the best 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.

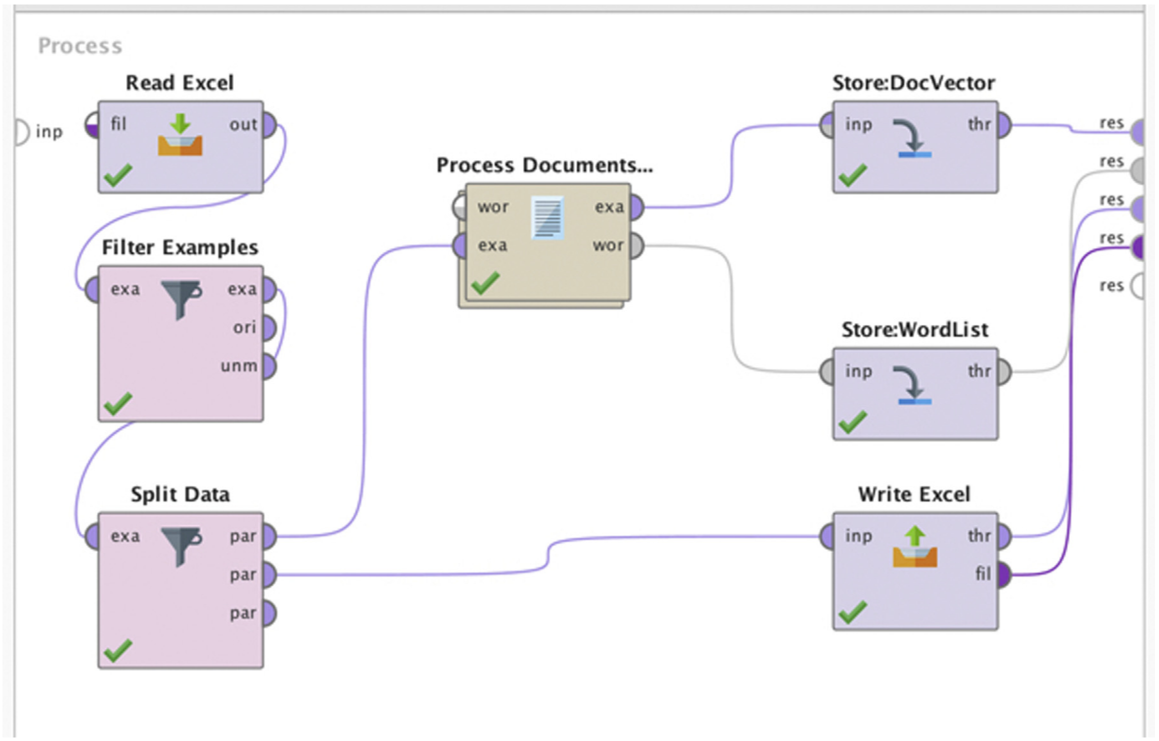


FIGURE 9.13
Preprocessing text data using the process documents from data operator.

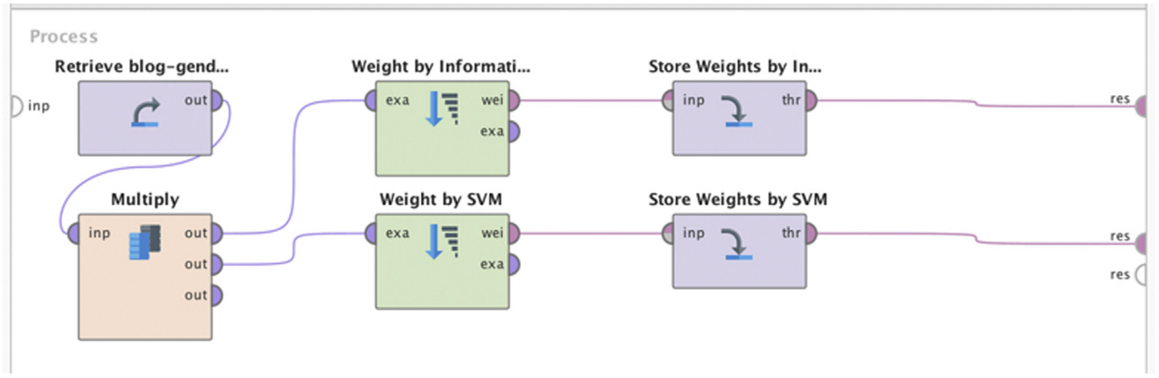


FIGURE 9.14
Using feature selection methods to filter attributes from the TDM. *TDM*, Term Document Matrix.

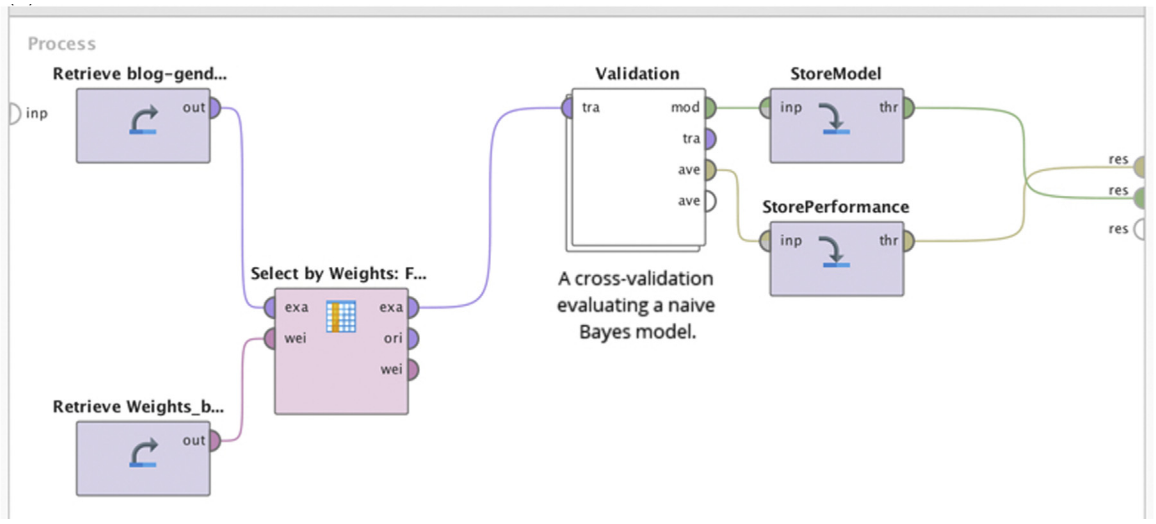


FIGURE 9.15
Training and testing predictive models for blog gender classification.

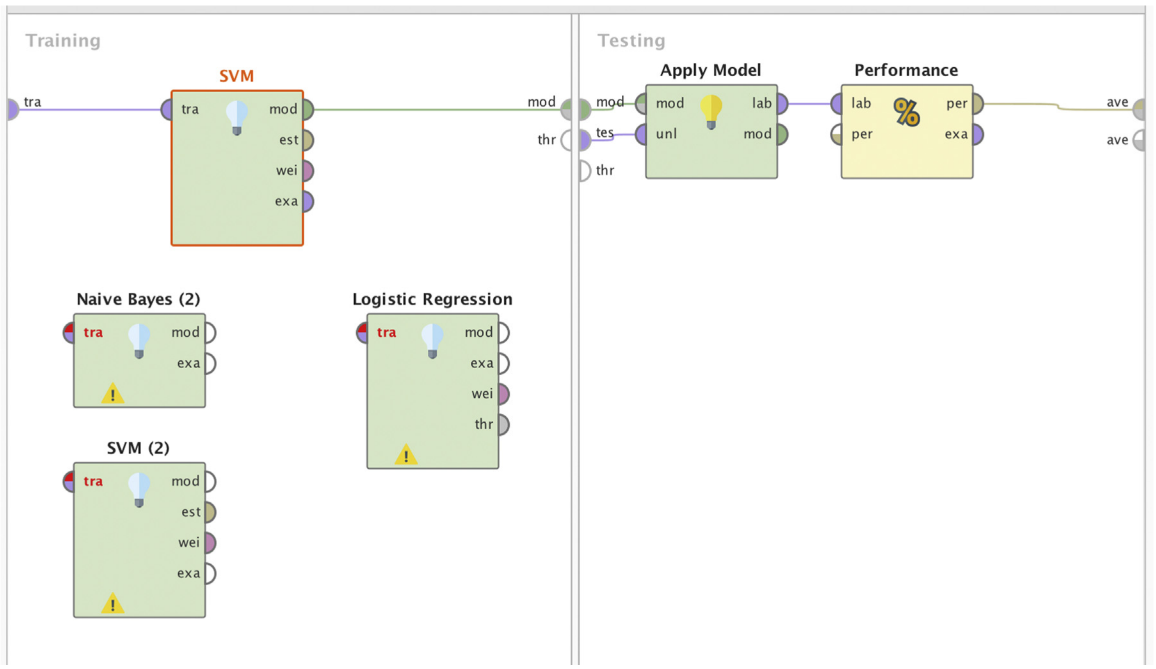


FIGURE 9.16
Switching between several algorithms.

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

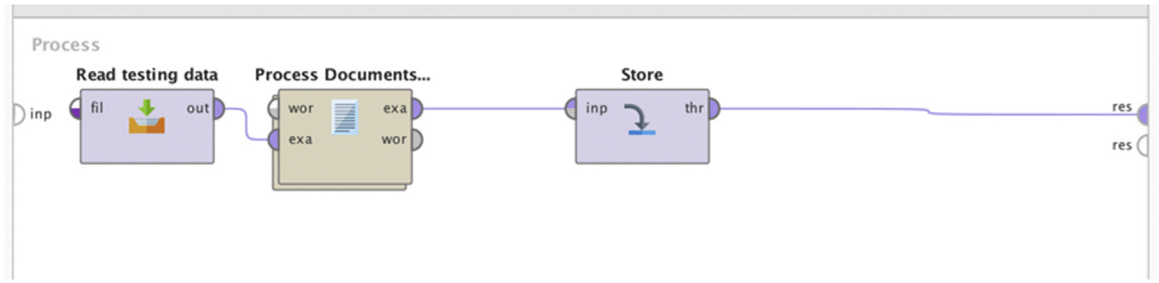


FIGURE 9.17
Preparing the unseen data for model deployment.

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 15: Getting started with RapidMiner, 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, the real-world performance of the best algorithm can actually be evaluated in classifying blogs by author gender. However, keep in mind that the raw data that was set aside as is, cannot be used (what would happen if one did?). This raw test data would also need to be converted into a document vector first. In other words, the step 2 process needs to be repeated (without the filtering and split data operators) on the 50% of the data that were set aside for testing. The *Process Documents from Data* operator can be simply copied and pasted from the process in step 2. (Alternatively, the entire data set could have been preprocessed before splitting!) The document vector is stored for use in the next step. This process is illustrated in Fig. 9.17.

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 in Figs. 9.18 and 9.19. 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 the results to be sorted from the Apply Model by Correct Predictions and Wrong Predictions using the View Filter in the Results perspective as shown here.

When this process is run, it can be observed that the LibSVM (linear) model can correctly predict only 828 of the 1468 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 optimization operators, one can easily improve on this baseline accuracy. A discussion about how to use the optimization operators in general is provided in Chapter 15, 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! Although this implementation did not return a stellar predictive performance (and the motivation of this classification is digital advertisement, as specified in the paper), a word of caution is in order: algorithms have become increasingly more powerful and reckless application of machine learning may lead to undesirable consequences of discrimination (via gender, in this instance). Data scientists are responsible for ensuring that their products are used in an ethical and non-discriminatory manner.

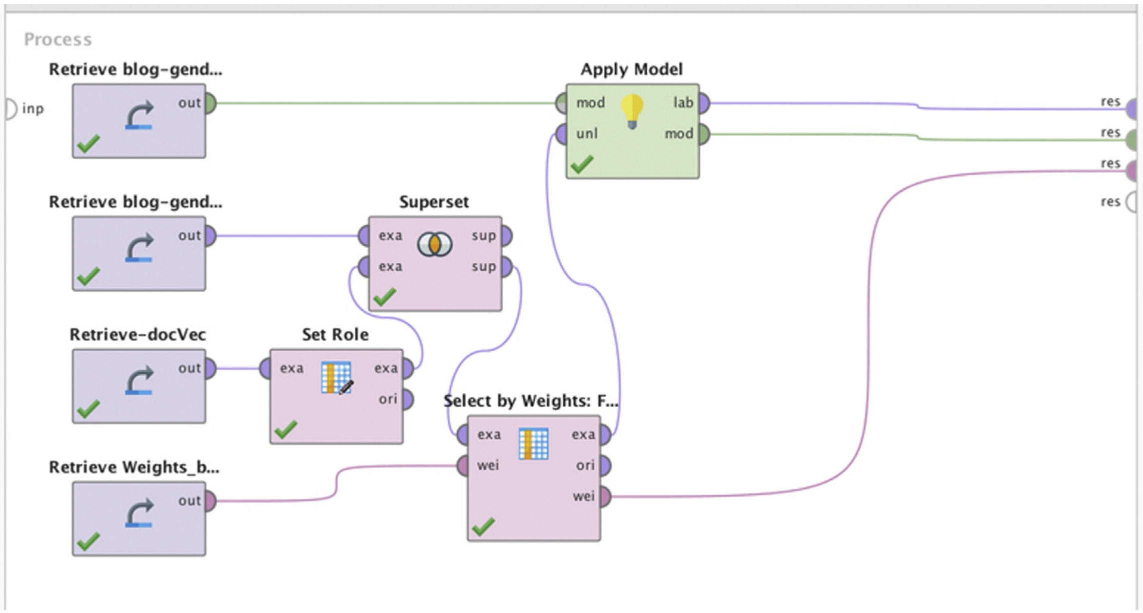
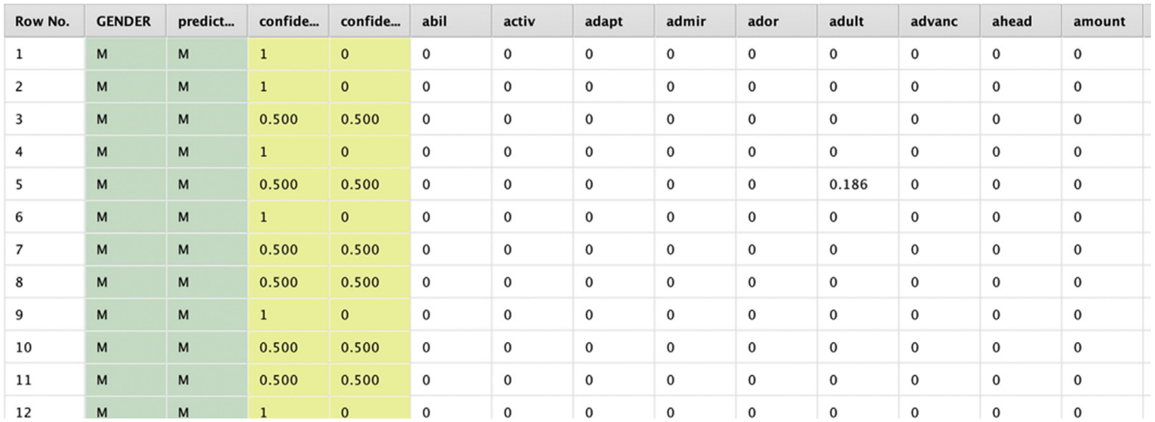


FIGURE 9.18
Applying the models built in step 3 on the unseen data.



Row No.	GENDER	predict...	confide...	confide...	abil	activ	adapt	admir	ador	adult	advanc	ahead	amount
1	M	M	1	0	0	0	0	0	0	0	0	0	0
2	M	M	1	0	0	0	0	0	0	0	0	0	0
3	M	M	0.500	0.500	0	0	0	0	0	0	0	0	0
4	M	M	1	0	0	0	0	0	0	0	0	0	0
5	M	M	0.500	0.500	0	0	0	0	0	0.186	0	0	0
6	M	M	1	0	0	0	0	0	0	0	0	0	0
7	M	M	0.500	0.500	0	0	0	0	0	0	0	0	0
8	M	M	0.500	0.500	0	0	0	0	0	0	0	0	0
9	M	M	1	0	0	0	0	0	0	0	0	0	0
10	M	M	0.500	0.500	0	0	0	0	0	0	0	0	0
11	M	M	0.500	0.500	0	0	0	0	0	0	0	0	0
12	M	M	1	0	0	0	0	0	0	0	0	0	0

FIGURE 9.19
The results view.

Bias in Machine Learning

Data Science is a powerful tool to extract value from data. Just like any other tool, it can be put to good use, inappropriate use, malicious use or use it in such a way that yield unintended consequences. Recently, a data science model that was developed to filter and sort the resumes of job applicants started to discriminate against women (Gershgorn, 2018). The data science modeling process would have started as a solution to a right business problem, which is to manage the influx of resumes and sorting the most relevant ones to the top. In doing so, the text mining model favoured one applicant class, leveraging some spurious pattern in the data. If the training data is biased, the machine learning model will be biased. Specifically, if the training data is derived from a biased process, the machine learning automation just amplifies the phenomenon. A loan approval model might not ask for the race of the applicant (which would be unethical and illegal). However, the location of an applicant might serve as a proxy for the race. It is important to test and audit if the model provides fair prediction for all the classes of users. These machine learning models have real world consequences. The responsibility lies with the data scientist to build a transparent model adhering to ethical principles. Creating robust tests to check for potential unfairness in the models is imperative to gain trust in the discipline of data science (Loukides et al., 2018).

9.3 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 one is now working on text documents and even voice recordings that have been transcribed to text.

This chapter described 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. A high-level three-step process that will enable this was introduced. Some key tools for transforming unstructured data, such as tokenization, stemming, *n*-gramming, and stop word removal were discussed. How concepts such as TF–IDF allow us to make the final transformation of a corpus of text to a matrix of numbers, that can be worked on by the standard machine learning algorithms was explained. Finally, a couple of implementation examples were presented, which will allow one to explore the exciting world of text mining.

References

Cutting, D. K. (1992). Scatter/gather: A cluster-based approach to browsing large document collections. In: *Copenhagen proceedings of the 15th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 318–329). Copenhagen.

Franz, A. A. (2006, August 3). *All our N-gram are belong to you*. Research blog. Retrieved from <http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html> Accessed 01.11.13.

Gershgorn, D. (2018). Companies are on the hook if their hiring algorithms are biased – Quartz. Retrieved October 24, 2018, from <https://qz.com/1427621/companies-are-on-the-hook-if-their-hiring-algorithms-are-biased/>.

Hearst, M. (1999, June 20–26). Untangling text data mining. In: *Proceedings of Association for Computational Linguistics, 37th annual meeting 1999*. University of Maryland.

Loukides, M., Mason, H., & Patil, D. J. (2018). *Ethics and Data Science*. Sebastopol. CA: O'Reilly Media.

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*. Information Management. Retrieved from <http://www.information-management.com/issues/20050101/1016487-1.html#Login> Accessed 01.11.13.

Mukherjee, A. L. (2010). Improving gender classification of blog authors. In: *Proceedings of conference on Empirical Methods in Natural Language Processing (EMNLP-10)*. Cambridge, MA.

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.

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*. Retrieved from <https://www.forbes.com/sites/bruceupbin/2013/02/08/ibms-watson-gets-its-first-piece-of-business-in-healthcare/#7010113b5402/>.