Chapter 9
Text Analytics

Sudhir Voleti

1 Introduction

The main focus of this textbook thus far has been the analysis of numerical data. Text analytics, introduced in this chapter, concerns itself with understanding and examining data in word formats, which tend to be more unstructured and therefore more complex. Text analytics uses tools such as those embedded in R in order to extract meaning from large amounts of word-based data. Two methods are described in this chapter: bag-of-words and natural language processing (NLP). This chapter is focused on the bag-of-words approach. The bag-of-words approach does not attribute meaning to the sequence of words. Its applications include clustering or segmentation of documents and sentiment analysis. Natural language processing uses the order and "type" of words to infer the meaning. Hence, NLP deals more with issues such as parts of speech.

2 Motivating Text Analysis

Consider the following scenarios: A manager wants to know the broad contours of what customers are saying when calling into the company's call center. A firm wants to know if there are persistent patterns in the content of their customer feedback

Electronic supplementary material The online version of this chapter (https://doi.org/10.1007/978-3-319-68837-4_9) contains supplementary material, which is available to authorized users.

S. Voleti (✉)
Indian School of Business, Hyderabad, Telangana, India
e-mail: sudhir_voleti@isb.edu

B. Pochiraju, S. Seshadri (eds.), *Essentials of Business Analytics*, International
Series in Operations Research & Management Science 264,
https://doi.org/10.1007/978-3-319-68837-4_9

records, customer complaints, or customer service calls/emails. An investor wants to know what major topics surround press coverage of a particular company. All these cases entail analyzing unstructured, open-ended text data, which may not be amenable to measurement and scaling along any of the four primary data scales. By several estimates, the vast majority of data flooding into organizations is unstructured, and much of this unstructured data is textual in form. Multiple customer touch-points in firms today, such as the call transcripts of a call center; email to customer service departments; social media outreach (Facebook comments, tweets, blog entries); speech transcripts, conference proceedings, press articles, statutory filings by friendly and rival firms; notes by field agents, salespeople, insurance inspectors, auditors; open-ended questions in direct interviews, surveys, typically yield unstructured data. Thus, there appears to be no getting away from the analysis of text data, especially in the field of business analytics. There is vast potential to unlocking sizeable value and competitive advantage.

2.1 An Illustrative Example: Google Flu Detector

Consider an illustrative example of text analysis from a non-business scenario. In 2009, a new flu virus was discovered, named H1N1. This virus spread quickly and there was no ready vaccine to deploy. Since it was potentially contagious, the best-case scenario was that the identified H1N1 cases would restrict their movements to stay at home avoiding contact with others. The US government had limited options—the primary agency dealing with the H1N1 threats, the Center for Disease Control (CDC) in Atlanta, had an information deficit. CDC's numbers came from doctors nationwide at the county level and were only updated every 1–2 weeks. But in that time, uninformed and unaware patients could accidentally spread H1N1 to even more people. It was at this stage that Google approached the CDC. Google reasoned that anyone with flu-like symptoms would likely search for it online and would probably use Google to do so. Moreover, they could pinpoint the origination of these searches for symptoms corresponding to H1N1 with great accuracy (right down to the street, house, hour and minute level). But to sort the wheat from the chaff, Google would need data on symptoms unique to H1N1, symptoms common to the flu, and most importantly—confirmed cases of flu in years past. The idea was to train a machine using data comprising years 2003–2008 to know which of the searched symptoms corresponded to those found in actual confirmed flu cases, and thereafter predict who might have the flu and/or H1N1 in the year going forward. The famous Google Flu detector was well received but later got embroiled in controversy and challenges regarding its prediction claims (par for the course in academia).

2.2 Data Sources for Text Mining

So what text data sources are individuals and organizations finding most common or useful for their various purposes? Figure 9.1 shows the results of a survey conducted at the 2014 Journal of Data Analysis Techniques (JDAT) conference in Europe, attended by academia and industry alike. The results are unsurprising. Social media and other user-generated content sources top the list—microblogs, full-length blogs, online forums, Facebook postings, etc. followed by more "traditional" text content sources such as mainstream media news articles, surveys (of customers, employees), reports, and medical records and compliance filings.

Data collection and extraction from these sources is now possible at scale by the development of new tools, standards, protocols, and techniques (e.g., Application Programming Interfaces (APIs)).

What are the building blocks of text analysis? What is the basic unit of analysis? How similar or different is text analysis from the analysis of more structured, metric data? The next section offers a briefing.

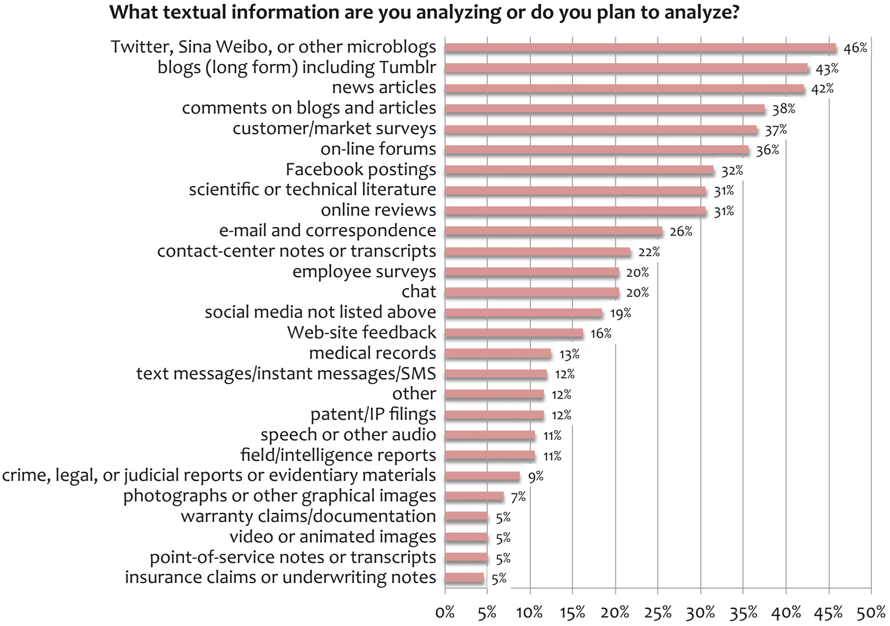
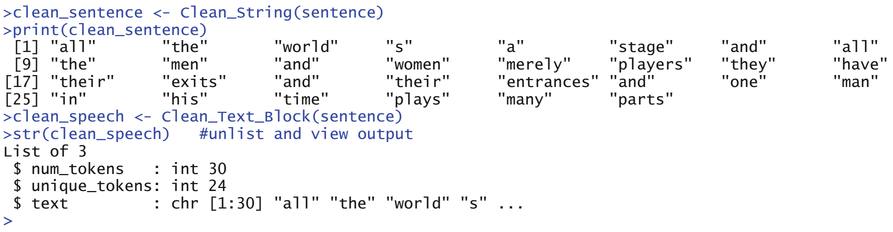


Fig. 9.1 Common sources for text data

3 Methods of Text Analysis

There are two broad approaches to handling text analysis. The first, bag-of-words, assumes that words in the text are “exchangeable,” that is, their order does not matter in conveying meaning. While this assumption vastly oversimplifies the quirks of text as a means of communication, it vastly reduces the dimensionality of the text object and makes the analysis problem very tractable. Thus, if the order did not matter then two occurrences of the same word in any text document could be clustered together and their higher-level summaries (such as counts and frequencies) could be used for analysis. Indeed, the starting point of most text analysis is a data object called the term-document matrix (TDM) which lists the counts for each term (word/phrase token) in each document within a given set of text documents. The second approach, natural language processing (NLP), attempts to interpret “natural language” and assumes that content (as also context) depends on the order and “type” of words used. Hence, NLP deals more with issues such as parts of speech and named entity recognition.

Consider a passage from Shakespeare’s *As You Like It*, “All the world’s a stage, and all the men and women merely players: they have their exits and their entrances; and one man in his time plays many parts . . . ” While some may see profound meaning in this (admittedly nonrandom) collection of words, Fig. 9.2 displays what happens when we process this text input in a computer’s text analysis program (in this case, the open source R platform, in particular its “*tm*” and “*stringr*” packages). The code that runs the above processing uses two user-defined functions, “*Clean_String()*” and “*Clean_Text_Block()*”, for which a tutorial can be found on Matt Denny’s Academic Website[1] or you may refer to the code (Fig_9.2_Shakespeare_quote.R) available on the book’s website. The breaking up of text into “words” (or more technically, word “tokens”) is called *tokenization*. While what we see in this simple routine are single-word tokens (or unigrams), it is possible to define, identify, and extract phrases of two or more words that “go



```
>clean_sentence <- Clean_String(sentence)
>print(clean_sentence)
 [1] "all"       "the"       "world"     "s"         "a"         "stage"     "and"       "all"
 [9] "the"       "men"       "and"       "women"     "merely"    "players"   "they"      "have"
[17] "their"     "exits"     "and"       "their"     "entrances" "and"       "one"       "man"
[25] "in"        "his"       "time"      "plays"     "many"      "parts"
>clean_speech <- Clean_Text_Block(sentence)
>str(clean_speech)   #unlist and view output
List of 3
 $ num_tokens   : int 30
 $ unique_tokens: int 24
 $ text         : chr [1:30] "all" "the" "world" "s" ...
>
```

Fig. 9.2 Basic text processing and unigram tokenization

[1] Tutorial link—http://www.mjdenny.com/Text_Processing_In_R.html (accessed on Dec 27, 2017).

together" in the text (bigrams, trigrams, etc.). However, that would require us to consider the order in which the words first occurred in the text and would form a part of NLP.

3.1 Some Terminology

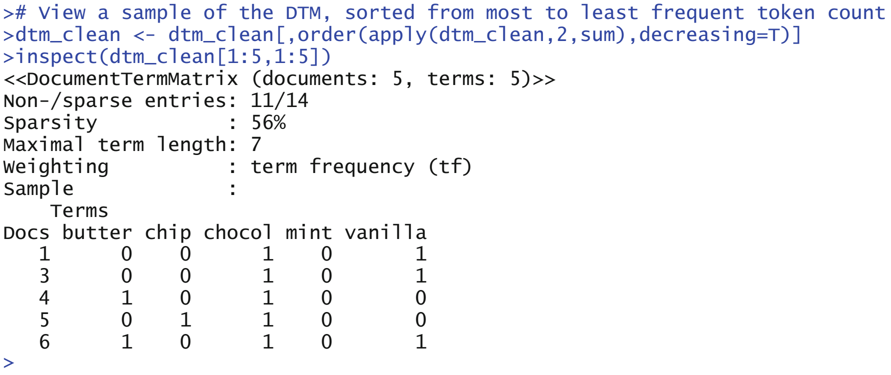
To introduce some terminology commonly associated with text analysis, let us consider a sample dataset called the "ice-cream"[2] dataset. It contains survey responses from over 5000 consumers to an online survey run by a mid-sized, regional US retail chain that wanted feedback and opinion on its soon-to-be-launched line of "light" store-branded ice-creams. Let us suppose the store-brand name is "*Wows*." Download the dataset from the website and save as a text file on your local machine in the working directory.

The responses are to one particular survey question of interest, "If *Wows* offered a line of light ice-creams, what flavors would you want to see? Please be as specific as possible." Each row (including the empty rows) in the dataset is a document, and the stack of documents is a text corpus. The following are some observations made after a cursory run through the first few sets of responses. Notice the blank rows (i.e., empty documents)—these are consumers who chose not to answer this open-ended question. Also notice the quirks of language and grammar. There are terms with typos in them (chocolate spelled without the second "o" or vanilla spelled with a single "l," etc.). Sometimes, the same terms occur both in lower and uppercase format in different documents. There are filler words—grammar's scaffolding—such as connectors, pronouns ("all," "for," "of," "my," "to," etc.) which, in a bag-of-words world, may not make much sense and would likely swamp other words with their relatively high frequency. And quite often, punctuations are all over the place.

It may help to mitigate the effects of such quirks of language by "standardizing" words and word-forms using some general rules that apply to the vast majority of situations in everyday text (though by no means, all of them). Thus, we could consider dropping the filler words, converting all the remaining terms to lowercase to avoid case-sensitivity effects, remove all special characters (numbers, punctuations, etc.), and "stem" the remaining words. To understand the process of stemming, think of each word as a tree and different variations of that word (borne by prefixes and suffixes) as branches. Stemming cuts out the branches and retains only the core "stem" of the word. Thus, for instance, the stem-word "run" replaces "runs," "running," etc.

As mentioned previously, tokenization is the process of breaking up a cleaned corpus into individual terms composed of 1-, 2-, or more word phrases. For example, "ice-cream" is a 2-word token (bigram), whereas ice and cream are 1-word tokens

[2] The dataset "icecream.csv" can be downloaded from the book's website.



```
># View a sample of the DTM, sorted from most to least frequent token count
>dtm_clean <- dtm_clean[,order(apply(dtm_clean,2,sum),decreasing=T)]
>inspect(dtm_clean[1:5,1:5])
<<DocumentTermMatrix (documents: 5, terms: 5)>>
Non-/sparse entries: 11/14
Sparsity           : 56%
Maximal term length: 7
Weighting          : term frequency (tf)
Sample             :
    Terms
Docs butter chip chocol mint vanilla
   1      0    0      1    0       1
   3      0    0      1    0       1
   4      1    0      1    0       0
   5      0    1      1    0       0
   6      1    0      1    0       1
>
```

Fig. 9.3 First few cells in a document term matrix in R

(unigrams). After a corpus is tokenized, a simple frequency table can be built to determine how many times each token occurred in each document. This frequency table is called the term-document matrix (TDM). It is one of the basic objects of analysis in text analytics. Each cell in a TDM records the frequency of a particular token in a particular document. TDMs tend to be large, sparse matrices. The TDM for the ice-cream dataset has a dimension of 907 (tokens) × 2213 (docs). Figure 9.3 shows the first few cells for the first few documents of a TDM for the ice-cream dataset, in order of the most frequently occurring tokens (listed left to right). Despite this, we find 56% sparsity (or, proportion of empty cells). Notice also that empty rows have been dropped. Notice that "chocolate-chip" and "chocolate" are two distinct tokens; the first is a bigram and the second a unigram. The code "icre" is available on the book's website for reference.

Having obtained a TDM, the next logical step is to display the simplest yet meaningful parts of the information contained in the TDM in a form that people can easily read and digest. One such output display avenue is the "wordcloud," which displays the most frequently occurring tokens in the corpus as a mass or cloud of words. The higher a word's frequency in the corpus, the larger its font size. Aside from font size, no other feature of the wordcloud—color, orientation, etc.—is informative. Figure 9.4a displays the wordcloud of the most frequently occurring tokens in the ice-cream dataset. The prevalence of bigrams can be seen. One can control how many tokens to show here, to avoid clutter.

A document-term matrix (DTM) is the transpose of a TDM, with documents as rows and tokens as columns, but is sometimes used interchangeably with TDM. There are generally two ways one can "weigh" a DTM. The first uses simple term frequency (or TF in short) wherein each column merely records token frequency per document. The issue with this method is that the token frequency may not necessarily imply token importance. For instance, in a dataset about opinions on ice-cream flavors, the bigram "ice-cream" is likely to occur with high frequency since it is expected to show up in most documents. However, its importance in relation to understanding customers' flavor preferences is relatively low. Hence, a

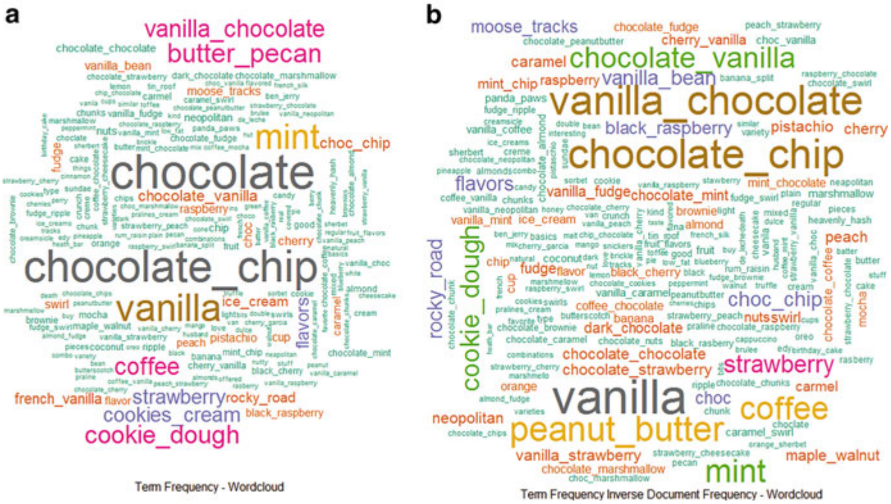


Fig. 9.4 (**a**, **b**) Wordclouds under the TF and TFIDF weighing schemes

second weighing scheme for tokens across documents, labeled TFIDF for "term frequency–inverse document frequency," has gained popularity. The basic idea is that a token's frequency should be normalized by the average number of times that token occurs in a document (i.e., document frequency). Thus, tokens with very high frequency spread evenly across documents tend to get discounted whereas those which occur more in some documents rather than all over the corpus gain in importance. Thus, "butter-scotch" would get a higher TFIDF score than "ice-cream" because only a subset of people would say the former, thereby raising its inverse document frequency. Various TFIDF weighing schemes have been proposed and implemented in different text analysis packages. It is prudent to test a few to see if they make more sense in the context of a given corpus than the simple TF scheme. Figure 9.4b displays a TFIDF wordcloud. The differences between that and the TF wordcloud in Fig. 9.4a are visibly apparent.

3.2 Co-occurrence Graphs (COG)

Wordclouds are very basic in that they can only say so much. Beyond simple term frequency, one might also want to know which tokens occur most frequently *together* within a document. For instance, do "vanilla" and "chocolate" go together? Or do "vanilla" and "peanut butter" go together more? More generally, does someone who uses the term "big data" also say "analytics"? A co-occurrence graph (COG) highlights token-pairs that tend to co-occur the most within documents across the corpus. The idea behind the COG is straightforward. If two tokens co-occur in documents more often than by random chance, we can use a network graph

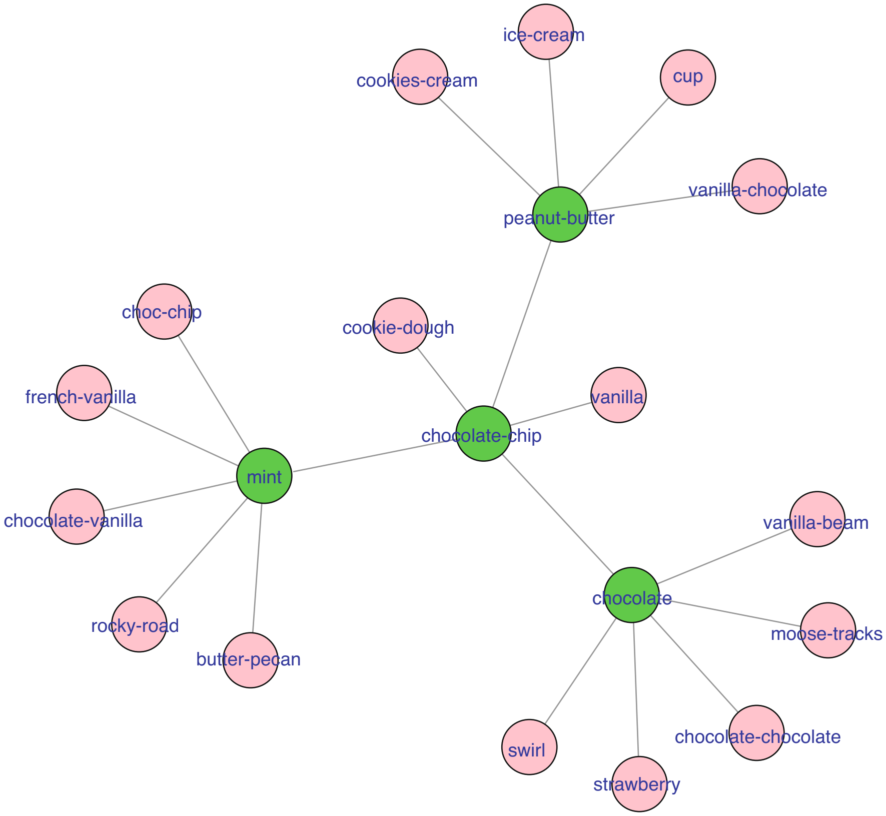


Fig. 9.5 A cleaned co-occurrence graph (COG) for the ice-cream dataset

framework to "connect" the two tokens as nodes with a link or an "edge." Because the odds are high that any two tokens will co-occur at least once in some document in the corpus, we introduce connection "thresholds" that ensure two node tokens are linked only if they co-occur more than the threshold number of times. Even so, we often see too many connections across too many nodes. Hence, in "cleaned" COGs, we designate nodes as "primary" (central to the graph) and "secondary" (peripheral nodes which only connect to one central node at a time), and we suppress interconnections between peripheral nodes for visual clarity. The R *shiny* app (discussed in the next section) can be used to generate a COG for the ice-cream dataset as shown in Fig. 9.5. Figure 9.5 displays one such cleaned COG that arises from the top tokens in the ice-cream dataset. We can interpret the COG in Fig. 9.5 as follows. We assume the peripheral (pink) nodes connect to one another only through the central (green) nodes, thus taking away much of the clutter in the earlier COG. Again, the links or edges between nodes appear only if the co-occurrence score crosses a predefined threshold.

3.3 Running Elementary Text Analysis

Today, a wide variety of platforms and tools are available to run standard text analysis routines. To run the analysis and obtain outputs similar to the ones shown in the figures, do the following.

- Open RStudio and copy-paste the code for "*shiny*" example[3] from author's github account into a script file.
  - Now, select lines 37–39 (#Basic Text-An-app) in RStudio and click "Run." RStudio first installs required libraries from the source file (line 38). Then it launches the *shiny* app for Factor An (line 39).
- Examine what the app is like—the Input Sidebar and the Output Tabs. This entire process has been described in the YouTube video tutorial[4] for a sample dataset and set of *shiny* apps in R.
- Now use the app and read in the ice-cream data, either as a text or a csv file.
- Once the app has run (may take up to several minutes to finish processing depending on the size of the dataset and the specification of the local machine), explore the output tabs to see what shows up. The video tutorial provides some assistance in this regard.

3.4 Elementary Text Analysis Applications

A number of interesting downstream applications become available once we have reduced the text corpus to an analysis object—the TDM. In the rest of this section, we discuss two such applications: (1) clustering or segmentation of documents based on text content and (2) elementary sentiment analysis.

Clustering Documents Using Text Bases

Clustering has had a long and interesting history in analytics, particularly in business where marketers are keen users of clustering methods under the broad rubric of "segmentation." However, while many applications in the past have relied on metric variables in datasets, we now examine a scenario where text data could become the basis for clustering documents. For our illustrative example, consider a marketing problem. Conceptually, segmentation in marketing implies the process of grouping together customers who share certain characteristics of interest, into homogenous segments. The rationale for segmentation is that it is more effective and efficient to pitch a value proposition to a relatively homogeneous segment than

[3] https://raw.githubusercontent.com/sudhir-voleti/profile-script/master/sudhir%20shiny%20app%20run%20lists.txt (accessed on Dec 27, 2017).

[4] https://www.youtube.com/watch?v=tN6FYIOe0bs (accessed on Dec 27, 2017) Sudhir Voleti is the creator of video.

to a heterogeneous mass. So, how do marketers segment and on what basis? Well, several bases can be considered: demographic (age, income, family size, zip code, etc.) and psychographic (brand conscious, price sensitive, etc.) are two examples. The ideal basis for marketers specifically is "customer need," which is often latent and hard to observe. In such instances, text offers a way to access qualitative insights that would otherwise not be reachable by traditional means. After segmentation, marketers would typically evaluate which segments are worth pursuing and how to target them.

Let us run a segmentation of the ice-cream survey's respondents based on the ice-cream flavor preferences they have freely stated in unstructured text form. The idea is to simply take the DTM and run the k-means clustering algorithm (see the Chap. 15 on Unsupervised Learning) on it. Thus, documents that are "similar" based on the tokens used would group together. To proceed with using the k-means algorithm, we must first input the number of clusters we want to work with.

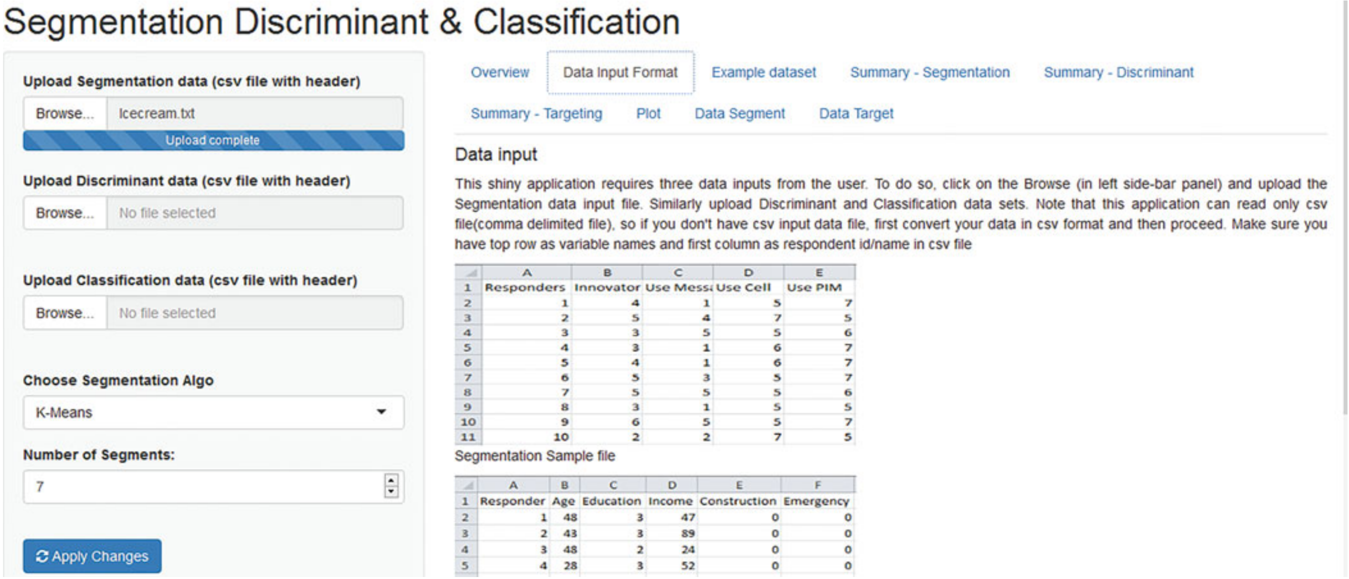
To run segmentation, select codes (#Segmentation-discriminant-targeting App) in RStudio and click "Run." The line 27 launches the *shiny* app for segmentation, discriminant, and classification. Upload the ice-cream data for segmentation (refer Fig 9.6). We can estimate the number of clusters through the *scree plot* on the "Summary—Segmentation" tab. Looking at where the "elbow" is sharpest (refer Fig. 9.7), we can arrive at what a reasonable number of clusters would be. The current example suggests that 2, 4, or 7 clusters might be optimal. Let us go with 7 (since we have over a thousand respondents).

We can interpret the output as follows. The "Summary—Segmentation" tab yields segment sizes, centroid profiles, and other classic descriptive data about the obtained clusters. The tab "Segmentation—Data" yields a downloadable version of segment assignment to each document. The two output tabs "Segmentation—Wordcloud" and "Segmentation co-occurrence" display segment wordclouds and COGs alongside segment sizes. We can observe that the rather large Segment 1 (at 44% of the sample) seems to be those who prefer vanilla followed by those who prefer butter-pecan at Segment 2 (8.4% of the sample) and so on.

Sentiment Analysis

Sentiment mining is an attempt to detect, extract, and assess value judgments, subjective opinion, and emotional content in text data. This raises the next question, "How is sentiment measured?" Valence is the technical term for the subjective inclination of a document—measured along a positive/neutral/negative continuum. Valence can be measured and scored.

Machines cannot determine the existence of sentiment content in a given word or phrase without human involvement. To aid machines in detecting and evaluating sentiment in large text corpora, we build word-lists of words and phrases that are likely to carry sentiment content. In addition, we attach a "score" to each word or phrase to represent the amount of sentiment content found in that text value. Since sentiment is typically measured along a positive/neutral/negative continuum, the steps needed to perform an analysis are fairly straightforward. First, we start with a processed and cleaned corpus. Second, we match the stem of each document in the corpus with positive (or negative) word-lists. Third, we score each document in the



Segmentation Discriminant & Classification

Upload Segmentation data (csv file with header)

Icecream.txt

Upload complete

Upload Discriminant data (csv file with header)

No file selected

Upload Classification data (csv file with header)

No file selected

Choose Segmentation Algo

K-Means

Number of Segments:

7

Data input

This shiny application requires three data inputs from the user. To do so, click on the Browse (in left side-bar panel) and upload the Segmentation data input file. Similarly upload Discriminant and Classification data sets. Note that this application can read only csv file(comma delimited file), so if you don't have csv input data file, first convert your data in csv format and then proceed. Make sure you have top row as variable names and first column as respondent id/name in csv file

	A	B	C	D	E
1	Responders	Innovator	Use Mess	Use Cell	Use PIM
2	1	4	1	5	7
3	2	5	4	7	5
4	3	3	5	5	6
5	4	3	1	6	7
6	5	4	1	6	7
7	6	5	3	5	7
8	7	5	5	5	6
9	8	3	1	5	5
10	9	6	5	5	7
11	10	2	2	7	5

Segmentation Sample file

	A	B	C	D	E	F
1	Responder	Age	Education	Income	Construction	Emergency
2	1	48	3	47	0	0
3	2	43	3	89	0	0
4	3	48	2	24	0	0
5	4	28	3	52	0	0

Fig. 9.6 Shiny app for segmentation and classification

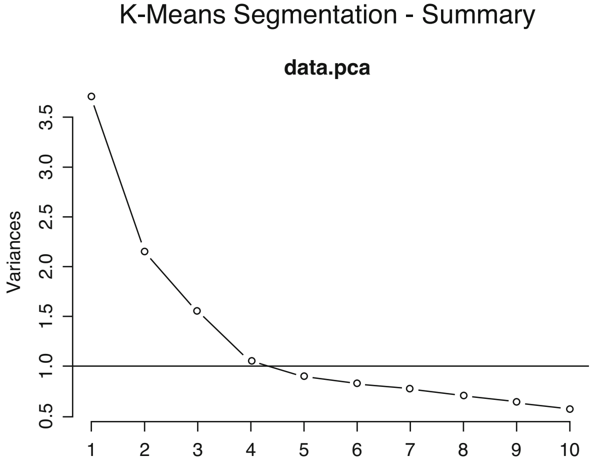


Fig. 9.7 K-means segmentation

corpus with a positive (or negative) "polarity." We can then plot the distribution of sentiment scores for each document as well as for the corpus as a whole. The positive and negative word-lists we use in the App are from the Princeton dictionary[5]. Since they are general, their effectiveness is somewhat limited in detecting sentiment in a given content. To customize sentiment analysis, one can and probably should build one's own context-specific sentiment scoring scheme giving valence weights for the most common phrases that occur in the domain of interest. Ideally, businesses and organizations would make domain-specific wordlists (and corresponding sentiment scores) for greater accuracy.

Let us see the result of running sentiment analysis on the ice-cream dataset. In the R app that you previously launched, look at the last two output tabs, namely "sentiment analysis" and "sentiment score data." One reason why the sentiment-laden wordclouds in the first output tab are so small is that most sentiment-laden tokens are adjectives while the ice-cream dataset consists of mostly nouns. Once sentiment scores by document are obtained, a host of downstream analysis becomes possible. The documents can now be sorted based on their "polarity" toward the topic at hand (i.e., ice-cream flavors). Such analyses are useful to businesses studying brand mentions on social media, for instance. Running trackers of sentiment over time can also provide useful information to managers. What we saw in this app was elementary sentiment mining. More advanced versions of sentiment mining and polarity-scoring schemes, leveraging natural language processing or NLP (discussed in detail in the next section), can be performed.

Let us step back and look at what we have covered with elementary text analysis thus far. In a nutshell, we have been able to rapidly crunch through raw text input on a scalable level, reduce open-ended text to a finite dimensional object (TDM), apply

[5] http://wordnet.princeton.edu/ (accessed on Feb 7, 2018).

standard analysis techniques (k-means segmentation) to this object, and sense what might be the major preference groups and important attributes that emerged through our analysis. These techniques help us to achieve a state in which a survey method can be leveraged for business insight. Next, we leave behind "basic" text analysis and head into a somewhat more advanced version wherein we will uncover hidden structure beneath text data, that is, latent topic mining and modeling.

3.5 Topic Mining Text Corpora

We know that organizations have huge data stored in text form—and it is likely that people are looking for ways to extract "meaning" from these texts. Imagine files piled up on a desk—the pile represents a corpus and each individual file a "document." One may ask, can we "summarize" the text content of those files? If so, what might a summary look like? One answer is that it should contain broad commonalities in coherent, inter-related text content patterns that occur across the files in the pile. Thus, one way to view meaning is in the form of coherent, condensed "topics" or "themes" that underlie a body of text. In the past, automated analysis has relied on simple models that do not directly address themes or topics, leaving us to derive meaning through a manual analysis of the text corpus. An approach to detect and extract coherent "themes" in the text data has become popular. It is the basis for topic mining of text corpus described in detail below.

To illustrate what underlying themes might look like or mean in a text corpus context, we take a simple example. Consider a text corpus of 20 product reviews. Suppose there are two broad themes or topics in the structure of the corpus, "price" and "brand." Further, suppose that each document is a mixture of these two topics in different proportions. We can then argue that a document that talks about "price" 90% of the time, and "brand" 10%, should have nine times more "price" terms than "brand" terms. A topic model formalizes this intuition mathematically. The algorithm yields which tokens belong to which topics with what probability, and which documents "load" (have strong association, see below for an example) on which topics with what proportion. Given this, we can sort, order, plot, analyze, etc. the tokens and the documents.

So, how can we directly mine for these latent topics or themes in text? The basic idea is analogous to the well-known Factor Analytic procedures (see the Chap. 15 on Unsupervised Learning). Recall what happens in traditional factor analysis. A dataset with R rows and C columns is factorized into two components—an R × F scores matrix and an F × C loadings matrix (R stands for the number of observations, C stands for the attributes of each observation, F then stands for the number of factors into which the data is decomposed). These factors are then labeled and interpreted in terms of the composite combinations of variables that load on them. For example, we can characterize each factor by observing which variables "load" highest onto it (Variable 1 positively, Variable 4 negatively, etc.). Using this we interpret what this means and give each factor an informative label.

Now let us see what would happen if instead of a traditional metric variable dataset, our dataset was a document term matrix (DTM) with D documents and T terms? And instead of conventional factors, we use the term “Topic Factors” for the factor. This would result in a D × F scores matrix of documents-on-factors, and an F × T loadings matrix of terms-on-factors. What are these matrices? What do these scores and loadings mean in a text context? Let us find out through a hands-on example.

The next dataset we will use is a comma-separated values (CSV) file containing the mission statements from a subset of Fortune 1000 firms. It is currently stored on the author’s github page. You can run the following lines of R code in RStudio to save the file on your local machine.

```
> # saving the data file as a .csv in your local machine
> mission.stments.data = read.csv(“https://raw.githubusercontent.
  com/sudhir-voleti/sample-data-
  sets/master/Mission%20Statements%20v1.csv”)
> # save file as data1.csv on your local machine
> write.csv(mission.stments.data, “data1.csv”)
> ### Topic Mining App
> source(“https://raw.githubusercontent.com/sudhir-voleti/text-
  topic-analysis-shinyapp/master/dependency-text-topic-analysis-
  shinyapp.R”)
> runGitHub(“text-topic-analysis-shinyapp”, “sudhir-voleti”)
```

Now invoke the *Topic mining* app in the *shiny* apps list (code above) and explore the app’s input fields and output tabs. Now read in the saved .csv file on mission statements into the app. Similar to the process used in our basic text analysis app, this too tokenizes the corpus, creates a TDM as its basic unit of analysis, and applies a latent topic model upon it. It calls upon the user (in this case, us) to tell it the optimal number of topics we think there are in the corpus. The default in the app is 2, but this can be manually changed.

The “TDM & Word Cloud” tab shows the corpus level wordcloud and the “Topic Model—Summary” tab shows the top few phrases loading onto each topic factor. The next three tabs display topic model output. At first glance, the output in the “Topics Wordcloud” and “Topics Co-occurrence” tabs may seem no different from those in the Basic Text An app. However, while the Basic Text An app represented TDM-based clusters (wherein entire documents are hard-allocated to one cluster or another), what we see now are topic factors that can co-exist within the same document. To recap, this model says in a nutshell that every document in the corpus is a mixture of topics in some proportion and that each topic is a distribution over word or phrase tokens with some topic membership probability. Both the topic membership probability vector per token and topic proportions vector per document can be simultaneously estimated in a Bayesian manner.

Returning to the example at hand, Fig. 9.8a, b show the wordclouds from Topics 1 and 2 respectively. Topic 1 seems to emphasize “corporation” and allied terms (services, systems, provide), whereas Topic 2 emphasizes “Customers” with allied terms (solutions, employees, product-services). This suggests that Topic 1 is perhaps a measure of company (and perhaps, product) centricity whereas Topic 2 is that of customer centricity in firms’ mission statements. As before, the co-occurrence graphs are used in conjunction with the wordclouds for better interpretation.

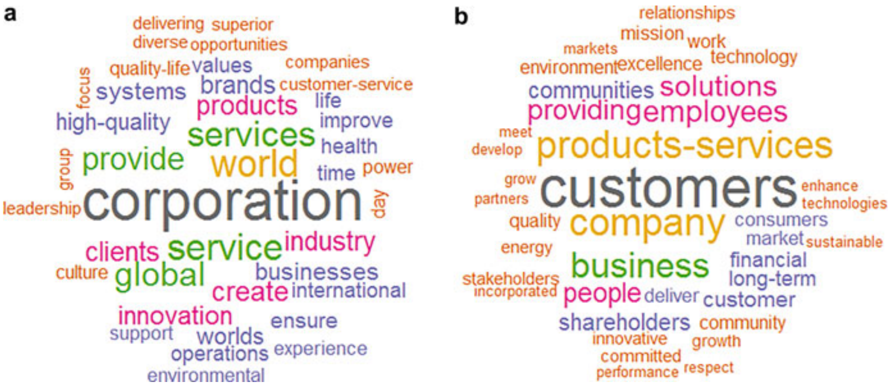


Fig. 9.8 (**a**, **b**) Wordclouds for topic factors 1 and 2 for the mission statements dataset

Finally, the last tab in the app, "Data with topic proportions" yields a downloadable file that shows the proportion of each document that is dominated by tokens loading on a particular topic. Thus, in this example, Microsoft corporation seems to have a 70–30 split between the company and customer centricity of its mission statement while that of toymaker Mattel, Inc. is 15–85.

4 Natural Language Processing (NLP)

While the "bag-of-words" approach we have seen thus far is useful, it has severe limitations. Human language or "natural language" approach is much more complex than a bag-of-words approach. The same set of words used in a different order could produce a different meaning. Even the same set of words uttered in a different tone could have a different meaning. Often, what precedes or succeeds particular sentences or paragraphs can impact the contextual meaning of any set of words. This brings our discussion to the natural language processing or NLP. By definition, NLP is a set of techniques that enable computers to detect nuances in human language that humans are able to detect automatically. Here, "nuances" refers to entities, relationships, context, and meaning among other things. So, what are some things we humans process automatically when reading or writing "natural" text? We parse text out into paragraphs and sentences—while we may not explicitly label the parts-of-speech (nouns, verbs, etc.), we can certainly understand and identify them. We notice names of people, places, dates, etc. ("entities") as they come up. And we can infer whether a sentence or paragraph portrays a happy, angry, or sad tone. What even human children appear to do effortlessly presents some tall challenges to the machine. Human language is too rich and subtle for computer languages to capture anywhere near the total amount of information "encoded" in it.

Between R and Python, the two main open-source data science alternatives, which is better for NLP? Python's natural language toolkit[6] (NLTK) is a clear winner here, but R is steadily closing the gaps with each passing quarter. The Apache OpenNLP[7] package in R provides access from the local machine to some trained NLP models. In what follows, let us very briefly see some of the main functions NLP of written text involves.

The simplest NLP functions involve recognizing sentences (and words) as distinct data forms. Tokenization does achieve some of this for words but sentence annotations come well within the ambit of NLP. So how does a machine identify that one sentence has ended and another has begun? Typically, large corpora of preclassified and annotated text content are fed to the machine and the machine trains off this data. The output of applying this trained algorithm on new or "virgin" data is an annotated document that delineates every word and sentence in the text separately. How can we use these sentence level annotations? Theoretically, sentences could now take the place of documents to act as our "rows" in the TDM. A simple sentence expresses a single idea, typically (unlike compound sentences or paragraphs). So, instead of doing sentiment analysis at the document level—we can do it at the sentence level. We can then see which associations (adjectives) appear with what nouns (brands, people, places, etc.). Co-occurrence graphs (COGs) can be built giving more weight to words co-occurring in sentences than in the document as a whole. We now have a building block that we could scale up and apply to documents and to corpora, in principle.

A popular application of NLP is in the Named Entity Recognition (NER) space. To illustrate, imagine a large pile of files on your work desk constituting of documents stacked into a corpus. Your task is to identify and extract every instance of a person's name, or an organization's name or phone-numbers or some combination of the above that occur in that corpus. This would then become an NER problem. An entity is basically a proper noun, such as the name of a person or place. In R, OpenNLP's NER annotator identifies and extracts entities of interest from an annotated document that we saw previously. OpenNLP can find dates, locations, money, organizations, percentages, people, and times (corresponding to "date," "location," "money," "organization," "percentage," "person," "misc"). The quality of the data recovery and the results that we get depend hugely on the type and training of the NER algorithm being employed. OpenNLP's NER annotator is fairly basic but does well on western entities. A more detailed exposition of NLP can be found in several excellent books on the subject, see for example Bird et al. (2009) and Robinson and Silge (2017). Also, Exercise 9.2 provides step-by-step guidance to NER problem-solving using OpenNLP package in R.

[6] NLTK package and documentation are available on http://www.nltk.org/ (accessed on Feb 10, 2018).

[7] Apache OpenNLP package and documentation are available on https://opennlp.apache.org/ (accessed on Feb 10, 2018).

5 Summary and Conclusion

Advances in information technology have enabled us to capture an ever-increasing amount of data both within and outside organizations, of which a significant portion is textual in form. For analysis, text data are organized into distinct "documents" that are stacked into a "corpus." The analysis of text data proceeds by first breaking down the text content into atomic form called "tokens"—basic units of analysis—that can easily be read into machines. Once a corpus has been tokenized, basic analysis structures such as term-document matrices (TDM) emerge which can form the building blocks of downstream analyses using linear algebra and econometric principles. We saw an overview of the kinds of basic analyses possible with text data: TDMs, display output through wordclouds and COGs, the use of clustering algorithms such as k-means upon a TDM to yield groups of documents that use similar token sets, the use of external wordlists to match tokens and give rise to weighted tokens (as happens in sentiment mining and analysis), etc.

We went further and explored latent topic mining in text corpora. The basic idea there was that TDMs could be factorized to yield latent text structure—coherent semantic themes that underlie and span the corpus—wherein each document is a composite of latent topics in some unknown but estimable proportion and each topic itself is a probability distribution over the token set. Finally, we saw a brief introduction to the quirks and challenges associated with directly mining spoken and written language as is, that is, natural language processing (NLP). The aim of this chapter is to provide an overview to business applications of such text data and we discuss through many examples as well as hands-on exercises with datasets and shiny apps in R (containing automated and interactivity enabled workflows) a way to do so.

Electronic Supplementary Material

All the datasets, code, and other material referred in this section are available in http://www.allaboutanalytics.net.

- Data 9.1: Generate_Document_Word_Matrix.cpp
- Code 9.1: Github_shiny_code.R
- Code 9.2: Icecream.R
- Data 9.2: Icecream.txt

Exercises

Ex. 9.1 Analyzing a simple set of documents.

Imagine you are a consultant for a movie studio. Your brief is to recommend the top 2–3 movie aspects or attributes the studio should focus on in making a sequel.

Go to IMDB and extract 100 reviews (50 positive and 50 negative) for your favorite movie.

(a) Preprocess the data like removing punctuation marks, numbers, ASCII characters, converting whole text to lowercase/uppercase, and removing stop-words and stemming.

Do elementary text analysis

(b) Create document term matrix.
(c) Check word-clouds and COGs under both TF (Term Frequency) and TFIDF (term frequency—inverse document frequency) weighing schemes for which configurations appear most meaningful/informative.
(d) Iterate by updating the stop-words list, etc.
(e) Compare each review's polarity score with its star rating. You can choose to use a simple cor() function to check correlation between the two data columns.
(f) Now, make a recommendation. What movie attributes or aspects (plot? star cast? length? etc.) worked well, which the studio should retain? Which ones did not work well and which the studio should change?

Explore with trial-and-error different configurations of possibilities (what stop-words to use for maximum meaning? TF or IDF? etc.) in the text analytics of a simple corpus. You may also use topic modeling if you wish.

Ex. 9.2 NLP for Entity recognition.

(a) Select one well-known firm from the list of the fortune 500 firms.
(b) For the selected firm, scrape its Wikipedia page.
(c) Using openNLP, find all the locations and persons mentioned in the Wikipedia page.

Note: You can use either openNLPs NER functionality, or, alternately, use the noun-phrase home-brewed chunker. If using the latter, manually separate persons and locations of interest.

(d) Plot all the extracted locations from the Wikipedia page on a map.
(e) Extract all references to numbers (dollar amounts, number of employees, etc.) using Regex.

Algorithm:

Step 1: Web scraping to choose one company among the list of Fortune 500 firms. For example, I have chosen Walmart.

Step 2: Navigate to the wiki page of the selected firm. For example: Copy-paste the top wiki paragraphs into a string in R.

Step 3: Install OpenNLP in R

1. Load the required packages in R and perform basic tokenization.
2. Now generate an annotator which will compute sentence and word annotations in openNLP.
3. Now, Annotate Persons and locations using the entity annotator. Example: Maxent_Entity_Annotator(Kind = "location") and Maxent_Entity_Annotator(Kind = "person").

Step 4: Load ggmap, rworldmap package to plot the locations in the map.

Step 5: Using regular expressions in R, match the patterns of numbers and display the results.

Ex. 9.3 Empirical topic modeling.

(a) Choose three completely different subjects. For example, choose “cricket, “macroeconomics,” and “astronomy.”

(b) Scrape Wikipedia pages related to the given subjects. Make each paragraph as a document and annotate each document for its respective category (approx. 50 paragraph should be analyzed for each category). For example on subject cricket we can search One Day International, test cricket, IPL etc.

(c) Now create a simulated corpus of 50 documents thus: The first of the 50 documents is a simple concatenation of the first document from subject 1, from subject 2 and from subject 3. Likewise, for the other 49 documents.

Thus, our simulated corpus now has “composite” documents, that is, documents composed of three distinct subjects each.

(d) Run the latent topic model code for k = 3 topics on this simulated corpus of 50 composite documents.

(e) Analyze the topic model results—Word clouds, COGs, topic proportions in documents.

(f) See

- Whether the topic model is able to separate each subject from other subjects. To what extent is it able to do so?
- Are there mixed tokens (with high lift in more than one topic)? Are the highest LIFT tokens and the document topic proportions (ETA scores) clear and able to identify each topic?

References

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python*. Sebastopol, CA: O’Reilly Media.

Robinson, D., & Silge, J. (2017). *Text mining with R: A tidy approach*. Sebastopol, CA: O’Reilly Media.