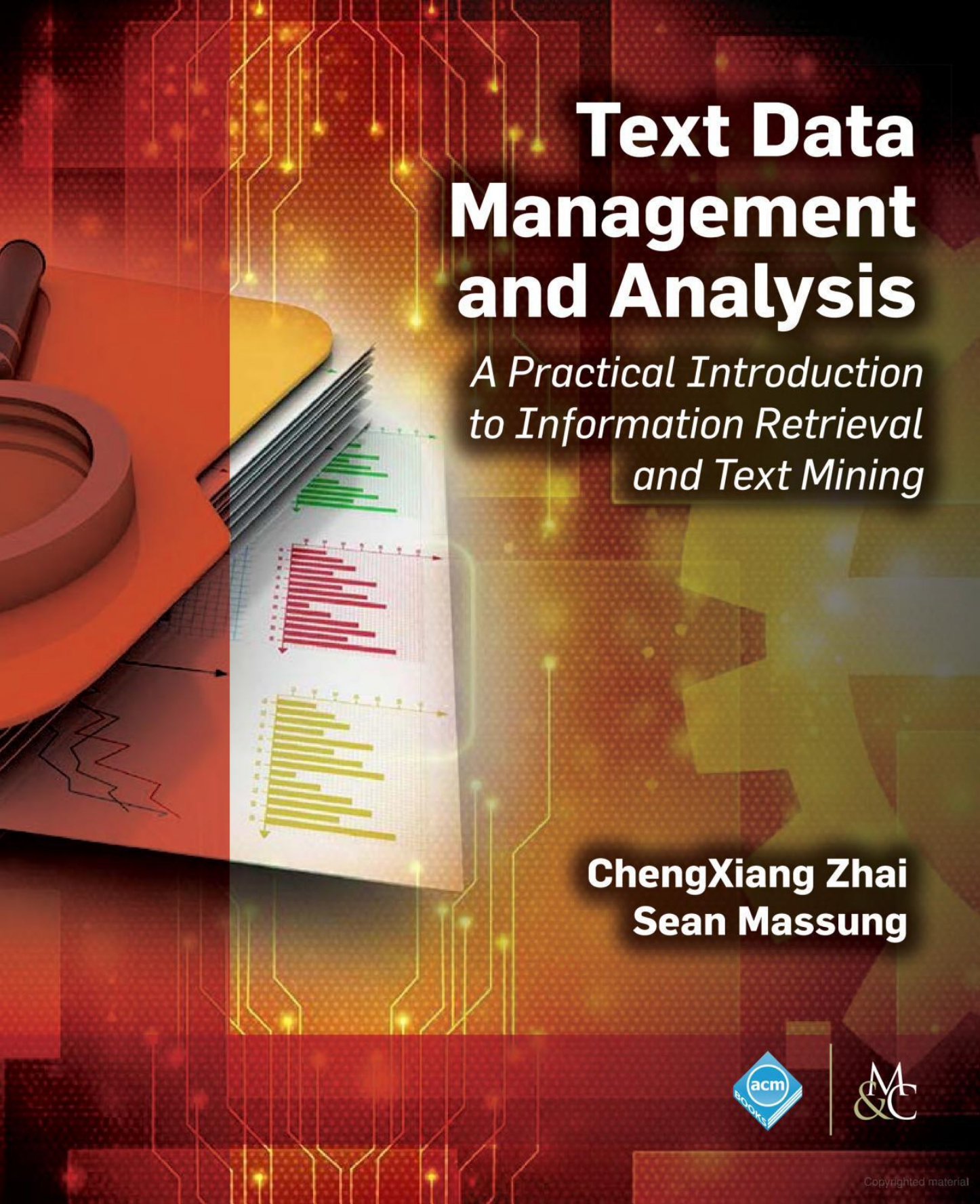

Text Data Management and Analysis

A Practical Introduction to Information Retrieval and Text Mining

ChengXiang Zhai

University of Illinois at Urbana–Champaign

Sean Massung

University of Illinois at Urbana–Champaign

ACM Books #12

Text Data Management and Analysis

ChengXiang Zhai and Sean Massung

books.acm.org
www.morganclaypoolpublishers.com

ISBN: 978-1-97000-119-8 hardcover
ISBN: 978-1-97000-116-7 paperback
ISBN: 978-1-97000-117-4 ebook
ISBN: 978-1-97000-118-1 ePub

Series ISSN: 2374-6769 print 2374-6777 electronic

DOIs:

10.1145/2915031 Book
10.1145/2915031.2915032 Preface
10.1145/2915031.2915033 Chapter 1
10.1145/2915031.2915034 Chapter 2
10.1145/2915031.2915035 Chapter 3
10.1145/2915031.2915036 Chapter 4
10.1145/2915031.2915037 Chapter 5
10.1145/2915031.2915038 Chapter 6
10.1145/2915031.2915039 Chapter 7
10.1145/2915031.2915040 Chapter 8
10.1145/2915031.2915041 Chapter 9
10.1145/2915031.2915042 Chapter 10
10.1145/2915031.2915043 Chapter 11
10.1145/2915031.2915044 Chapter 12
10.1145/2915031.2915045 Chapter 13
10.1145/2915031.2915046 Chapter 14
10.1145/2915031.2915047 Chapter 15
10.1145/2915031.2915048 Chapter 16
10.1145/2915031.2915049 Chapter 17
10.1145/2915031.2915050 Chapter 18
10.1145/2915031.2915051 Chapter 19
10.1145/2915031.2915052 Chapter 20
10.1145/2915031.2915053 Appendices
10.1145/2915031.2915054 References
10.1145/2915031.2915055 Index

A publication in the ACM Books series, #12
Editor in Chief: M. Tamer Özsu, *University of Waterloo*
Area Editor: Edward A. Fox, *Virginia Tech*

First Edition

10 9 8 7 6 5 4 3 2 1

Contents

Preface

The growth of "big data" created unprecedented opportunities to leverage computational and statistical approaches to turn raw data into actionable knowledge that can support various application tasks. This is especially true for the optimization of decision making in virtually all application domains such as health and medicine, security and safety, learning and education, scientific discovery, and business intelligence. Just as a microscope enables us to see things in the "micro world" and a telescope allows us to see things far away, one can imagine a "big data scope" would enable us to extend our perception ability to "see" useful hidden information and knowledge buried in the data, which can help make predictions and improve the optimality of a chosen decision. This book covers general computational techniques for managing and analyzing large amounts of text data that can help users manage and make use of text data in all kinds of applications.

Text data include all data in the form of natural language text (e.g., English text or Chinese text): all the web pages, social media data such as tweets, news, scientific literature, emails, government documents, and many other kinds of enterprise data. Text data play an essential role in our lives. Since we communicate using natural languages, we produce and consume a large amount of text data every day on all kinds of topics. The explosive growth of text data makes it impossible, or at least very difficult, for people to consume all the relevant text data in a timely manner. Thus, there is an urgent need for developing intelligent information retrieval systems to help people manage the text data and get access to the needed relevant information quickly and accurately at any time. This need is a major reason behind the recent growth of the web search engine industry. Due to the fact that text data are produced by humans for communication purposes, they are generally rich in semantic content and often contain valuable knowledge, information, opinions, and preferences of people. Thus, as a special kind of "big data," text data offer a great opportunity to discover various kinds of knowledge useful for many applications, especially knowledge about human opinions and preferences, which is often

directly expressed in text data. For example, it is now the norm for people to tap into opinionated text data such as product reviews, forum discussions, and social media text to obtain opinions. Once again, due to the overwhelming amount of information, people need intelligent software tools to help discover relevant knowledge for optimizing decisions or helping them complete their tasks more efficiently. While the technology for supporting text mining is not yet as mature as search engines for supporting text access, significant progress has been made in this area in recent years, and specialized text mining tools have now been widely used in many application domains. The subtitle of this book suggests that we cover two major topics, *information retrieval* and *text mining*. These two topics roughly correspond to the techniques needed to build the two types of application systems discussed above (i.e., search engines and text analytics systems), although the separation of the two is mostly artificial and only meant to help provide a high-level structure for the book, and a sophisticated application system likely would use many techniques from both topic areas.

In contrast to structured data, which conform to well-defined schemas and are thus relatively easy for computers to handle, text has less explicit structure so the development of intelligent software tools discussed above requires computer processing to understand the content encoded in text. The current technology of natural language processing has not yet reached a point to enable a computer to precisely understand natural language text (a main reason why humans often should be involved in the loop), but a wide range of statistical and heuristic approaches to management and analysis of text data have been developed over the past few decades. They are usually very robust and can be applied to analyze and manage text data in any natural language, and about any topic. This book intends to provide a systematic introduction to many of these approaches, with an emphasis on covering the most useful knowledge and skills required to build a variety of practically useful text information systems.

This book is primarily based on the materials that the authors have used for teaching a course on the topic of text data management and analysis (i.e., CS410 Text Information Systems) at the University of Illinois at Urbana–Champaign, as well as the two Massive Open Online Courses (MOOCs) on "Text Retrieval and Search Engines" and "Text Mining and Analytics" taught by the first author on Coursera in 2015. Most of the materials in the book directly match those of these two MOOCs with also similar structures of topics. As such, the book can be used as a main reference book for any of these two MOOCs.

Information Retrieval (IR) is a relatively mature field and there are no shortage of good textbooks on IR; for example, the most recent ones include *Modern Information Retrieval: The Concepts and Technology behind Search* by Baeza-Yates

and Ribeiro-Neto [2011], *Information Retrieval: Implementing and Evaluating Search Engines* by Büttcher et al. [2010], *Search Engines: Information Retrieval in Practice* by Croft et al. [2009], and *Introduction to Information Retrieval* by Manning et al. [2008]. Compared with these existing books on information retrieval, our book has a broader coverage of topics as it attempts to cover topics in both information retrieval and text mining, and attempts to paint a general roadmap for building a text information system that can support both text information access and text analysis. For example, it includes a detailed introduction to word association mining, probabilistic topic modeling, and joint analysis of text and non-text data, which are not available in any existing information retrieval books. In contrast with IR, Text Mining (TM) is far from mature and is actually still in its infancy. Indeed, how to define TM precisely remains an open question. As such, it appears that there is not yet a textbook on TM. As a textbook on TM, our book provides a basic introduction to the major representative techniques for TM. By introducing TM and IR in a unified framework, we want to emphasize the importance of integration of IR and TM in any practical text information system since IR plays two important roles in any TM application. The first is to enable fast reduction of the data size by filtering out a large amount of non-relevant text data to obtain a small set of most relevant data to a particular application problem. The second is to support an analyst to verify and interpret any patterns discovered from text data where an analyst would need to use search and browsing functions to reach and examine the most relevant support data to the pattern.

Another feature that sets this book apart is the availability of a companion toolkit for information retrieval and text mining, i.e., the MeTA toolkit (available at https://meta-toolkit.org/), which contains implementations of many techniques discussed in the book. Many exercises in the book are also designed based on this toolkit to help readers acquire practical skills of experimenting with the learned techniques from the book and applying them to solve real-world application problems.

This book consists of four parts. Part I provides an overview of the content covered in the book and some background knowledge needed to understand the chapters later. Parts II and III contain the major content of the book and cover a wide range of techniques in IR (called Text Data Access techniques) and techniques in TM (called Text Data Analysis techniques), respectively. Part IV summarizes the book with a unified framework for text management and analysis where many techniques of IR and TM can be combined to provide more advanced support for text data access and analysis with humans in the loop to control the workflow.

The required background knowledge to understand the content in this book is minimal since the book is intended to be mostly self-contained. However, readers

are expected to have basic knowledge about computer science, particularly data structures and programming languages and be comfortable with some basic concepts in probability and statistics such as conditional probability and parameter estimation. Readers who do not have this background may still be able to follow the basic ideas of most of the algorithms discussed in the book; they can also acquire the needed background by carefully studying Chapter 2 of the book and, if necessary, reading some of the references mentioned in the Bibliographical Notes section of that chapter to have a solid understanding of all the major concepts mentioned therein. MeTA can be used by anyone to easily experiment with algorithms and build applications, but modifying it or extending it would require at least some basic knowledge of C++ programming.

The book can be used as a textbook for an upper-level undergraduate course on information retrieval and text mining or a reference book for a graduate course to cover practical aspects of information retrieval and text mining. It should also be useful to practitioners in industry to help them acquire a wide range of practical techniques for managing and analyzing text data that they can use immediately to build various interesting real-world applications.

Acknowledgments

This book is the result of many people's help. First and foremost, we want to express our sincere thanks to Edward A. Fox for his invitation to write this book for the ACM Book Series in the area of Information Retrieval and Digital Libraries, of which he is the Area Editor. We are also grateful to Tamer Ozsu, Editor-in-Chief of ACM Books, for his support and useful comments on the book proposal. Without their encouragement and support this book would have not been possible. Next, we are deeply indebted to Edward A. Fox, Donna Harman, Bing Liu, and Jimmy Lin for thoroughly reviewing the initial draft of the book and providing very useful feedback and constructive suggestions. While we were not able to fully implement all their suggestions, all their reviews were extremely helpful and led to significant improvement of the quality of the book in many ways; naturally, any remaining errors in the book are solely the responsibility of the authors.

Throughout the process of writing the book, we received strong support and great help from Diane Cerra, Executive Editor at Morgan & Claypool Publishers, whose regular reminders and always timely support are key factors that prevented us from having the risk of taking "forever" to finish the book; for this, we are truly grateful to her. In addition, we would like to thank Sara Kreisman for copyediting and Paul C. Anagnostopoulos and his production team at Windfall Software (Ted

Laux, Laurel Muller, MaryEllen Oliver, and Jacqui Scarlott) for their great help with indexing, illustrations, art proofreading, and composition, which ensured a fast and smooth production of the book.

The content of the book and our understanding of the topics covered in the book have benefited from many discussions and interactions with a large number of people in both the research community and industry. Due to space limitations, we can only mention some of them here (and have to apologize to many whose names are not mentioned): James Allan, Charu Aggarwal, Ricardo Baeza-Yates, Nicholas J. Belkin, Andrei Broder, Jamie Callan, Jaime Carbonell, Kevin C. Chang, Yi Chang, Charlie Clarke, Fabio Crestani, W. Bruce Croft, Maarten de Rijke, Arjen de Vries, Daniel Diermeier, AnHai Doan, Susan Dumais, David A. Evans, Edward A. Fox, Ophir Frieder, Norbert Fuhr, Evgeniy Gabrilovich, C. Lee Giles, David Grossman, Jiawei Han, Donna Harman, Marti Hearst, Jimmy Huang, Rong Jin, Thorsten Joachims, Paul Kantor, David Karger, Diane Kelly, Ravi Kumar, Oren Kurland, John Lafferty, Victor Lavrenko, Lillian Lee, David Lewis, Jimmy Lin, Bing Liu, Wei-Ying Ma, Christopher Manning, Gary Marchionini, Andrew McCallum, Alistair Moffat, Jian-Yun Nie, Douglas Oard, Dragomir R. Radev, Prabhakar Raghavan, Stephen Robertson, Roni Rosenfeld, Dan Roth, Mark Sanderson, Bruce Schatz, Fabrizio Sebastiani, Amit Singhal, Keith van Rijsbergen, Luo Si, Noah Smith, Padhraic Smyth, Andrew Tomkins, Ellen Voorhees, and Yiming Yang, Yi Zhang, Justin Zobel. We want to thank all of them for their indirect contributions to this book. Some materials in the book, especially those in Chapter 19, are based on the research work done by many Ph.D. graduates of the Text Information Management and Analysis (TIMAN) group at the University of Illinois at Urbana–Champaign, under the supervision by the first author. We are grateful to all of them, including Tao Tao, Hui Fang, Xuehua Shen, Azadeh Shakery, Jing Jiang, Qiaozhu Mei, Xuanhui Wang, Bin Tan, Xu Ling, Younhee Ko, Alexander Kotov, Yue Lu, Maryam Karimzadehgan, Yuanhua Lv, Duo Zhang, V.G.Vinod Vydiswaran, Hyun Duk Kim, Kavita Ganesan, Parikshit Sondhi, Huizhong Duan, Yanen Li, Hongning Wang, Mingjie Qian, and Dae Hoon Park. The authors' own work included in the book has been supported by multiple funding sources, including NSF, NIH, NASA, IARPA, Air Force, ONR, DHS, Alfred P. Sloan Foundation, and many companies including Microsoft, Google, IBM, Yahoo!, LinkedIn, Intel, HP, and TCL. We are thankful to all of them.

The two Massive Open Online Courses (MOOCs) offered by the first author for the University of Illinois at Urbana–Champaign (UIUC) in 2015 on Coursera (i.e., *Text Retrieval and Search Engines* and *Text Mining and Analytics*) provided a direct basis for this book in the sense that many parts of the book are based primarily on the transcribed notes of the lectures in these two MOOCs. We thus would like

to thank all the people who have helped with these two MOOCs, especially TAs Hussein Hazimeh and Alex Morales, and UIUC instruction support staff Jason Mock, Shannon Bicknell, Katie Woodruff, and Edward Noel Dignan, and the Head of Computer Science Department, Rob Rutenbar, whose encouragement, support, and help are all essential for these two MOOCs to happen. The first author also wants to thank UIUC for allowing him to use the sabbatical leave in Fall 2015 to work on this book. Special thanks are due to Chase Geigle, co-founder of MeTA. In addition to all the above, the second author would like to thank Chase Geigle, Jason Cho, and Urvashi Khandelwal (among many others) for insightful discussion and encouragement.

Finally, we would like to thank all our family members, particularly our wives, Mei and Kai, for their love and support. The first author wants to further thank his brother Chengxing for the constant intellectual stimulation in their regular research discussions and his parents for cultivating his passion for learning and sharing knowledge with others.

ChengXiang Zhai
Sean Massung
June 2016

PART I

OVERVIEW AND BACKGROUND

1 Introduction

In the last two decades, we have experienced an explosive growth of online information. According to a study done at University of California Berkeley back in 2003: ". . . the world produces between 1 and 2 exabytes (1018 petabytes) of unique information per year, which is roughly 250 megabytes for every man, woman, and child on earth. Printed documents of all kinds comprise only .03% of the total." [Lyman et al. 2003]

A large amount of online information is textual information (i.e., in natural language text). For example, according to the Berkeley study cited above: "Newspapers represent 25 terabytes annually, magazines represent 10 terabytes . . . office documents represent 195 terabytes. It is estimated that 610 billion emails are sent each year representing 11,000 terabytes." Of course, there are also blog articles, forum posts, tweets, scientific literature, government documents, etc. Roe [2012] updates the email count from 610 billion emails in 2003 to 107 *trillion* emails sent in 2010. According to a recent IDC report report [Gantz & Reinsel 2012], from 2005 to 2020, the digital universe will grow by a factor of 300, from 130 exabytes to 40,000 exabytes, or 40 trillion gigabytes.

While, in general, all kinds of online information are useful, textual information plays an especially important role and is arguably the most useful kind of information for the following reasons.

Text (natural language) is the most natural way of encoding human knowledge. As a result, most human knowledge is encoded in the form of text data. For example, scientific knowledge almost exclusively exists in scientific literature, while technical manuals contain detailed explanations of how to operate devices.

Text is by far the most common type of information encountered by people. Indeed, most of the information a person produces and consumes daily is in text form.

Text is the most expressive form of information in the sense that it can be used to describe other media such as video or images. Indeed, image search engines such as those supported by Google and Bing often rely on matching companion text of images to retrieve "matching" images to a user's keyword query.

The explosive growth of online text information has created a strong demand for intelligent software tools to provide the following two related services to help people manage and exploit big text data.

Text Retrieval. The growth of text data makes it impossible for people to consume the data in a timely manner. Since text data encode much of our accumulated knowledge, they generally cannot be discarded, leading to, e.g., the accumulation of a large amount of literature data which is now beyond any individual's capacity to even skim over. The rapid growth of online text information also means that no one can possibly digest all the new information created on a daily basis. Thus, there is an urgent need for developing intelligent text retrieval systems to help people get access to the needed relevant information quickly and accurately, leading to the recent growth of the web search industry. Indeed, web search engines like Google and Bing are now an essential part of our daily life, serving millions of queries daily. In general, search engines are useful anywhere there is a relatively large amount of text data (e.g., desktop search, enterprise search or literature search in a specific domain such as PubMed).

Text Mining. Due to the fact that text data are produced by humans for communication purposes, they are generally rich in semantic content and often contain valuable knowledge, information, opinions, and preferences of people. As such, they offer great opportunity for discovering various kinds of knowledge useful for many applications, especially knowledge about human opinions and preferences, which is often directly expressed in text data. For example, it is now the norm for people to tap into opinionated text data such as product reviews, forum discussions, and social media text to obtain opinions about topics interesting to them and optimize various decision-making tasks such as purchasing a product or choosing a service. Once again, due to the overwhelming amount of information, people need intelligent software tools to help discover relevant knowledge to optimize decisions or help them complete their tasks more efficiently. While the technology for supporting text mining is not yet as mature as search engines for supporting text access, sig-

nificant progress has been made in this area in recent years, and specialized text mining tools have now been widely used in many application domains.

In contrast to structured data, which conform to well-defined schemas and are thus relatively easy for computers to handle, text has less explicit structure, so the development of intelligent software tools discussed above requires computer processing to understand the content encoded in text. The current technology of natural language processing has not yet reached a point to enable a computer to precisely understand natural language text (a main reason why humans often should be involved in the loop), but a wide range of statistical and heuristic approaches to management and analysis of text data have been developed over the past few decades. They are usually very robust and can be applied to analyze and manage text data in any natural language, and about any topic. This book intends to provide a systematic introduction to many of these approaches, with an emphasis on covering the most useful knowledge and skills required to build a variety of practically useful text information systems.

The two services discussed above (i.e., text retrieval and text mining) conceptually correspond to the two natural steps in the process of analyzing any "big text data" as shown in Figure 1.1. While the raw text data may be large, a specific application often requires only a small amount of most relevant text data, thus conceptually, the very first step in any application should be to identify the *relevant text data* to a particular application or decision-making problem and avoid the unnecessary processing of large amounts of non-relevant text data. This first step of converting the raw big text data into much smaller, but highly relevant text data is often accomplished by techniques of text retrieval with help from users (e.g., users may use multiple queries to collect all the relevant text data for a decision problem). In this first step, the main goal is to connect users (or applications) with the most relevant text data.

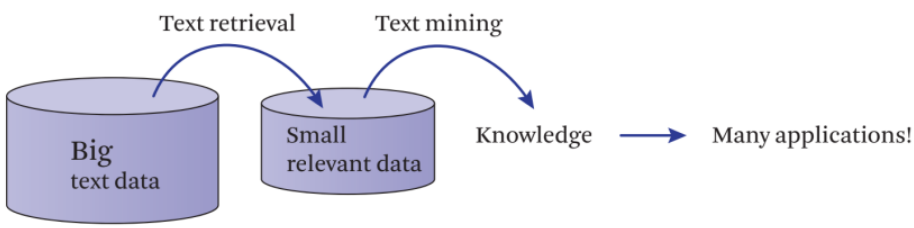

Figure 1.1 **Text retrieval and text mining are two main techniques for analyzing big text data.**

Once we obtain a small set of most relevant text data, we would need to further analyze the text data to help users digest the content and knowledge in the text data. This is the text mining step where the goal is to further discover knowledge and patterns from text data so as to support a user's task. Furthermore, due to the need for assessing trustworthiness of any discovered knowledge, users generally have a need to go back to the original raw text data to obtain appropriate context for interpreting the discovered knowledge and verify the trustworthiness of the knowledge, hence a search engine system, which is primarily useful for text access, also has to be available in any text-based decision-support system for supporting knowledge provenance. The two steps are thus conceptually interleaved, and a full-fledged intelligent text information system must integrate both in a unified framework.

It is worth pointing out that put in the context of "big data," text data is very different from other kinds of data because it is generally produced directly by humans and often also meant to be consumed by humans as well. In contrast, other data tend to be machine-generated data (e.g., data collected by using all kinds of physical sensors). Since humans can understand text data far better than computers can, involvement of humans in the process of mining and analyzing text data is absolutely crucial (much more necessary than in other big data applications), and how to optimally divide the work between humans and machines so as to optimize the collaboration between humans and machines and maximize their "combined intelligence" with minimum human effort is a general challenge in all applications of text data management and analysis. The two steps discussed above can be regarded as two different ways for a text information system to assist humans: information retrieval systems assist users in finding from a large collection of text data the most relevant text data that are actually needed for solving a specific application problem, thus effectively turning big raw text data into much smaller relevant text data that can be more easily processed by humans, while text mining application systems can assist users in analyzing patterns in text data to extract and discover useful actionable knowledge directly useful for task completion or decision making, thus providing more direct task support for users.

With this view, we partition the techniques covered in the book into two parts to match the two steps shown in Figure 1.1, which are then followed by one chapter to discuss how all the techniques may be integrated in a unified text information system. The book attempts to provide a complete coverage of all the major concepts, techniques, and ideas in information retrieval and text data mining from a practical viewpoint. It includes many hands-on exercises designed with a companion software toolkit MeTA to help readers learn how to apply techniques of information

retrieval and text mining to real-world text data and learn how to experiment with and improve some of the algorithms for interesting application tasks. This book can be used as a textbook for computer science undergraduates and graduates, library and information scientists, or as a reference book for practitioners working on relevant application problems in analyzing and managing text data.

1.1 Functions of Text Information Systems

From a user's perspective, a text information system (TIS) can offer three distinct, but related capabilities, as illustrated in Figure 1.2.

Information Access. This capability gives a user access to the useful information when the user needs it. With this capability, a TIS can connect the right information with the right user at the right time. For example, a search engine enables a user to access text information through querying, whereas a recommender system can push relevant information to a user as new information items become available. Since the main purpose of Information Access is to connect a user with relevant information, a TIS offering this capability

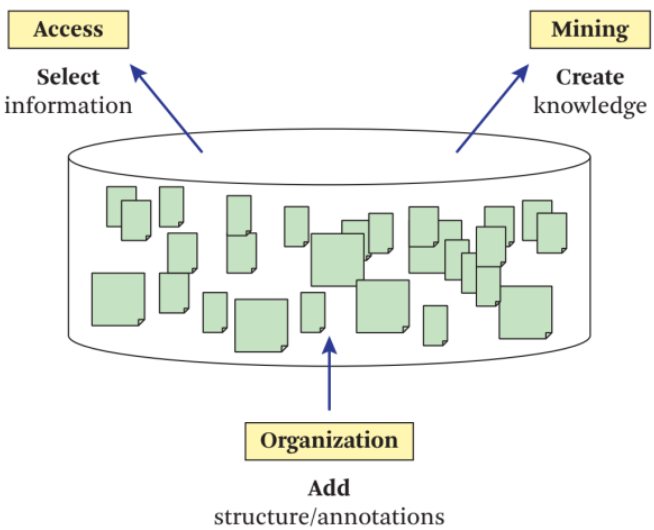

Figure 1.2 **Information access, knowledge acquisition, and text organization are three major capabilities of a text information system with text organization playing a supporting role for information access and knowledge acquisition. Knowledge acquisition is also often referred to as text mining.**

generally only does minimum analysis of text data sufficient for matching relevant information with a user's information need, and the original information items (e.g., web pages) are often delivered to the user in their original form, though summaries of the delivered items are often provided. From the perspective of text analysis, a user would generally need to read the information items to further digest and exploit the delivered information.

Knowledge Acquisition (Text Analysis). This capability enables a user to acquire useful knowledge encoded in the text data that is not easy for a user to obtain without synthesizing and analyzing a relatively large portion of the data. In this case, a TIS can analyze a large amount of text data to discover interesting patterns buried in text. A TIS with the capability of knowledge acquisition can be referred to as an analysis engine. For example, while a search engine can return relevant reviews of a product to a user, an analysis engine would enable a user to obtain directly the major positive or negative opinions about the product and to compare opinions about multiple similar products. A TIS offering the capability of knowledge acquisition generally would have to analyze text data in more detail and synthesize information from multiple text documents, discover interesting patterns, and create new information or knowledge.

Text Organization. This capability enables a TIS to annotate a collection of text documents with meaningful (topical) structures so that scattered information can be connected and a user can navigate in the information space by following the structures. While such structures may be regarded as "knowledge" acquired from the text data, and thus can be directly useful to users, in general, they are often only useful for facilitating either information access or knowledge acquisition, or both. In this sense, the capability of text organization plays a supporting role in a TIS to make information access and knowledge acquisition more effective. For example, the added structures can allow a user to search with constraints on structures or browse by following structures. The structures can also be leveraged to perform detailed analysis with consideration of constraints on structures.

Information access can be further classified into two modes: *pull* and *push*. In the pull mode, the user takes initiative to "pull" the useful information out from the system; in this case, the system plays a passive role and waits for a user to make a request, to which the system would then respond with relevant information. This mode of information access is often very useful when a user has an *ad hoc*

information need, i.e., a temporary information need (e.g., an immediate need for opinions about a product). For example, a search engine like Google generally serves a user in pull mode. In the push mode, the system takes initiative to "push" (recommend) to the user an information item that the system believes is useful to the user. The push mode often works well when the user has a relatively stable information need (e.g., hobby of a person); in such a case, a system can know "in advance" a user's preferences and interests, making it feasible to recommend information to a user without having the user to take the initiative. We cover both modes of information access in this book.

The pull mode further consists of two complementary ways for a user to obtain relevant information: *querying* and *browsing*. In the case of querying, the user specifies the information need with a (keyword) query, and the system would take the query as input and return documents that are estimated to be relevant to the query. In the case of browsing, the user simply navigates along structures that link information items together and progressively reaches relevant information. Since querying can also be regarded as a way to navigate, in one step, into a set of relevant documents, it's clear that browsing and querying can be interleaved naturally. Indeed, a user of a web search engine often interleaves querying and browsing.

Knowledge acquisition from text data is often achieved through the process of text mining, which can be defined as mining text data to discover useful knowledge. Both the data mining community and the natural language processing (NLP) community have developed methods for text mining, although the two communities tend to adopt slightly different perspective on the problem. From a data mining perspective, we may view text mining as mining a special kind of data, i.e., text. Following the general goals of data mining, the goal of text mining would naturally be regarded as to discover and extract interesting patterns in text data, which can include latent topics, topical trends, or outliers. From an NLP perspective, text mining can be regarded as to partially understand natural language text, convert text into some form of knowledge representation and make limited inferences based on the extracted knowledge. Thus a key task is to perform *information extraction*, which often aims to identify and extract mentions of various entities (e.g., people, organization, and location) and their relations (e.g., who met with whom). In practice, of course, any text mining applications would likely involve both pattern discovery (i.e., data mining view) and information extraction (i.e., NLP view), with information extraction serving as enriching the semantic representation of text, which enables pattern

finding algorithms to generate semantically more meaningful patterns than directly working on word or string-level representations of text. Due to our emphasis on covering general and robust techniques that can work for all kinds of text data without much manual effort, we mostly adopt the data mining view in this book since information extraction techniques tend to be more language-specific and generally require much manual effort. However, it is important to stress that information extraction is an essential component in any text information system that attempts to support deeper knowledge discovery or semantic analysis.

Applications of text mining can be classified as either direct applications, where the discovered knowledge would be directly consumed by users, or indirect applications, where the discovered knowledge isn't necessarily directly useful to a user, but can indirectly help a user through better support of information access. Knowledge acquisition can also be further classified based on what knowledge is to be discovered. However, due to the wide range of variations of the "knowledge," it is impossible to use a small number of categories to cover all the variations. Nevertheless, we can still identify a few common categories which we cover in this book. For example, one type of knowledge that a TIS can discover is a set of topics or subtopics buried in text data, which can serve as a concise summary of the major content in the text data. Another type of knowledge that can be acquired from opinionated text is the overall sentiment polarity of opinions about a topic.

1.2 Conceptual Framework for Text Information Systems

Conceptually, a text information system may consist of several modules, as illustrated in Figure 1.3.

First, there is a need for a module of *content analysis* based on natural language processing techniques. This module allows a TIS to transform raw text data into more meaningful representations that can be more effectively matched with a user's query in the case of a search engine, and more effectively processed in general in text analysis. Current NLP techniques mostly rely on *statistical machine learning* enhanced with limited linguistic knowledge with variable depth of understanding of text data; shallow techniques are robust, but deeper semantic analysis is only feasible for very limited domains. Some TIS capabilities (e.g., summarization) tend to require deeper NLP than others (e.g., search). Most text information systems use very shallow NLP, where text would simply be represented as a "*bag of words*," where words are basic units for representation and the order of words is ignored (although the counts of words are retained). However, a more sophisticated representation is

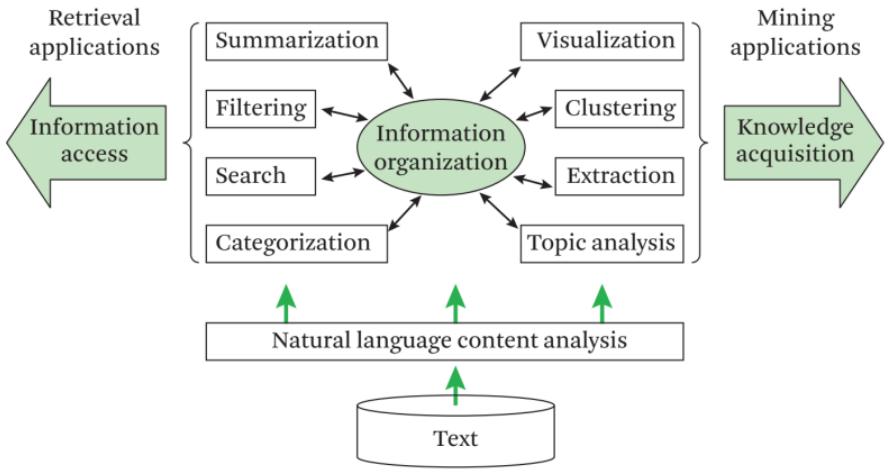

Figure 1.3 **Conceptual framework of text information systems.**

also possible, which may be based on recognized entities and relations or other techniques for more in-depth understanding of text.

With content analysis as the basis, there are multiple components in a TIS that are useful for users in different ways. The following are some commonly seen functions for managing and analyzing text information.

Search. Take a user's query and return relevant documents. The search component in a TIS is generally called a search engine. Web search engines are among the most useful search engines that enable users to effectively and efficiently deal with a huge amount of text data.

Filtering/Recommendation. Monitor an incoming stream, decide which items are relevant (or non-relevant) to a user's interest, and then recommend relevant items to the user (or filter out non-relevant items). Depending on whether the system focuses on recognizing relevant items or non-relevant items, this component in a TIS may be called a recommender system (whose goal is to recommend relevant items to users) or a filtering system (whose goal is to filter out non-relevant items to allow a user to keep only the relevant items). Literature recommender and spam email filter are examples of a recommender system and a filtering system, respectively.

Categorization. Classify a text object into one or several of the predefined categories where the categories can vary depending on applications. The categorization component in a TIS can annotate text objects with all kinds of meaningful categories, thus enriching the representation text data, which further enables more effective and deeper text analysis. The categories can also be used for organizing text data and facilitating text access. Subject categorizers that classify a text article into one or multiple subject categories and sentiment taggers that classify a sentence into positive, negative, or neutral in sentiment polarity are both specific examples of a text categorization system.

Summarization. Take one or multiple text documents, and generate a concise summary of the essential content. A summary reduces human effort in digesting text information and may also improve the efficiency in text mining. The summarization component of a TIS is called a summarizer. News summarizer and opinion summarizer are both examples of a summarizer.

Topic Analysis. Take a set of documents and extract and analyze topics in them. Topics directly facilitate digestion of text data by users and support browsing of text data. When combined with the companion non-textual data such as time, location, authors, and other meta data, topic analysis can generate many interesting patterns such as temporal trends of topics, spatiotemporal distributions of topics, and topic profiles of authors.

Information Extraction. Extract entities, relations of entities or other "knowledge nuggets" from text. The information extraction component of a TIS enables construction of entity-relation graphs. Such a knowledge graph is useful in multiple ways, including support of navigation (along edges and paths of the graph) and further application of graph mining algorithms to discover interesting entity-relation patterns.

Clustering. Discover groups of similar text objects (e.g., terms, sentences, documents, . . .). The clustering component of a TIS plays an important role in helping users explore an information space. It uses empirical data to create meaningful structures that can be useful for browsing text objects and obtaining a quick understanding of a large text data set. It is also useful for discovering outliers by identifying the items that do not form natural clusters with other items.

Visualization. Visually display patterns in text data. The visualization component is important for engaging humans in the process of discovering interesting patterns. Since humans are very good at recognizing visual patterns,

visualization of the results generated from various text mining algorithms is generally desirable.

This list also serves as an outline of the major topics to be covered later in this book. Specifically, search and filtering are covered first in Part II about text data access, whereas categorization, clustering, topic analysis, and summarization are covered later in Part III about text data analysis. Information extraction is not covered in this book since we want to focus on general approaches that can be readily applied to text data in *any* natural language, but information extraction often requires language-specific techniques. Visualization is also not covered due to the intended focus on algorithms in this book. However, it must be stressed that both information extraction and visualization are very important topics relevant to text data analysis and management. Readers interested in these techniques can find some useful references in the Bibliographic Notes at the end of this chapter.

1.3 Organization of the Book

The book is organized into four parts, as shown in Figure 1.4.

Part I. Overview and Background. This part consists of the first four chapters and provides an overview of the book and background knowledge, including basic concepts needed for understanding the content of the book that some readers may not be familiar with, and an introduction to the MeTA toolkit used for exercises in the book. This part also gives a brief overview of natural language processing techniques needed for understanding text data and obtaining informative representation of text needed in all text data analysis applications.

Part II. Text Data Access. This part consists of Chapters 5–11, covering the major techniques for supporting text data access. This part provides a systematic discussion of the basic information retrieval techniques, including the formulation of retrieval tasks as a problem of ranking documents for a query (Chapter 5), retrieval models that form the foundation of the design of ranking functions in a search engine (Chapter 6), feedback techniques (Chapter 7), implementation of retrieval systems (Chapter 8), and evaluation of retrieval systems (Chapter 9). It then covers web search engines, the most important application of information retrieval so far (Chapter 10), where techniques for analyzing links in text data for improving ranking of text objects are introduced and application of supervised machine learning to combine multiple

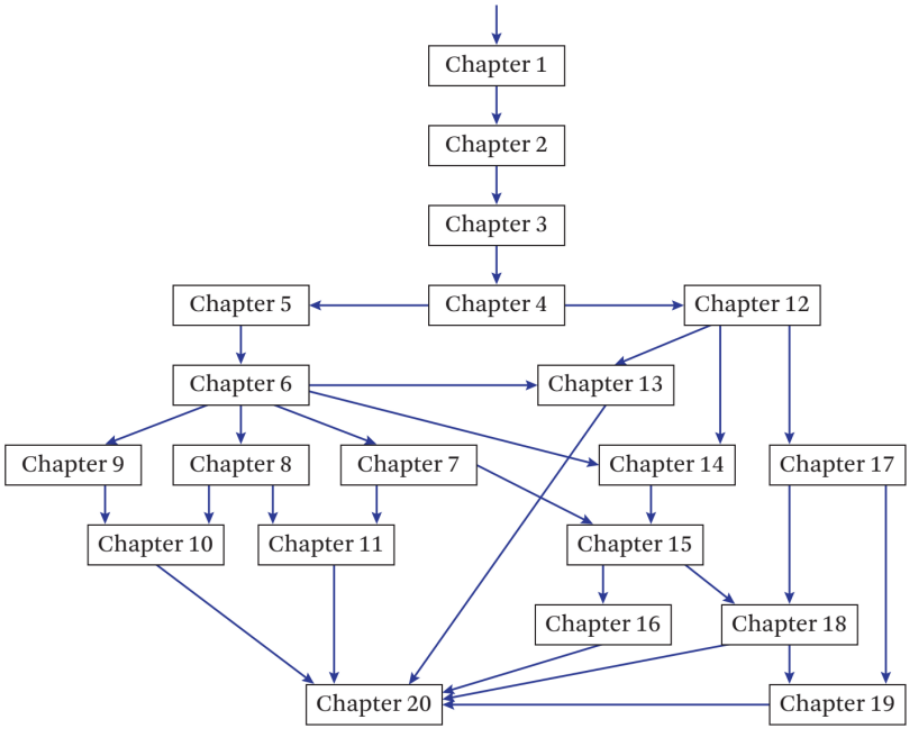

Figure 1.4 **Dependency relations among the chapters.**

features for ranking is briefly discussed. The last chapter in this part (Chapter 11) covers recommender systems which provide a "push" mode of information access, as opposed to the "pull" mode of information access supported by a typical search engine (i.e., querying by users).

Part III. Text Data Analysis. This part consists of Chapters 12–19, covering a variety of techniques for analyzing text data to facilitate user digestion of text data and discover useful topical or other semantic patterns in text data. Chapter 12 gives an overview of text analysis from the perspective of data mining, where we may view text data as data generated by humans as "subjective sensors" of the world; this view allows us to look at the text analysis problem in the more general context of data analysis and mining in general, and facilitates the discussion of joint analysis of text and non-text data. This is followed by multiple chapters covering a number of the most useful general techniques for analyzing text data without or with only minimum human effort. Specifically, Chapter 13 discusses techniques for discovering two fundamental se-

mantic relations between lexical units in text data, i.e., paradigmatic relations and syntagmatic relations, which can be regarded as an example of discovering knowledge about the natural language used to generate the text data (i.e., linguistic knowledge). Chapter 14 and Chapter 15 cover, respectively, two closely related techniques to generate and associate meaningful structures or annotations with otherwise unorganized text data, i.e., text clustering and text categorization. Chapter 16 discusses text summarization useful for facilitating human digestion of text information. Chapter 17 provides a detailed discussion of an important family of probabilistic approaches to discovery and analysis of topical patterns in text data (i.e., topic models). Chapter 18 discusses techniques for analyzing sentiment and opinions expressed in text data, which are key to discovery of knowledge about preferences, opinions, and behavior of people based on analyzing the text data produced by them. Finally, Chapter 19 discusses joint analysis of text and non-text data, which is often needed in many applications since it is in general beneficial to use as much data as possible for gaining knowledge and intelligence through (big) data analysis.

Part IV. Unified Text Management and Analysis System. This last part consists of Chapter 20 where we attempt to discuss how all the techniques discussed in this book can be conceptually integrated in an operator-based unified framework, and thus potentially implemented in a general unified system for text management and analysis that can be useful for supporting a wide range of different applications. This part also serves as a roadmap for further extension of MeTA to provide effective and general high-level support for various applications and provides guidance on how MeTA may be integrated with many other related existing toolkits, including particularly search engine systems, database systems, natural language processing toolkits, machine learning toolkits, and data mining toolkits.

Due to our attempt to treat all the topics from a practical perspective, most of the discussions of the concepts and techniques in the book are informal and intuitive. To satisfy the needs of some readers that might be interested in deeper understanding of some topics, the book also includes an appendix with notes to provide a more detailed and rigorous explanation of a few important topics.

1.4 How to Use this Book

Due to the extremely broad scope of the topics that we would like to cover, we have to make many tradeoffs between breadth and depth in coverage. When making

such a tradeoff, we have chosen to emphasize the coverage of the basic concepts and practical techniques of text data mining at the cost of not being able to cover many advanced techniques in detail, and provide some references at the end of many chapters to help readers learn more about those advanced techniques if they wish to. Our hope is that with the foundation received from reading this book, you will be able to learn about more advanced techniques by yourself or via another resource. We have also chosen to cover more general techniques for text management and analysis and favor techniques that can be applicable to any text in any natural language. Most techniques we discuss can be implemented without any human effort or only requiring minimal human effort; this is in contrast to some more detailed analysis of text data, particularly using natural language processing techniques. Such "deep analysis" techniques are obviously very important and are indeed necessary for some applications where we would like to go in-depth to understand text in detail. However, at this point, these techniques are often not scalable and they tend to require a large amount of human effort. In practice, it would be beneficial to combine both kinds of techniques.

We envision three main (and potentially overlapping) categories of readers.

Students. This book is specifically designed to give you hands-on experience in working with real text mining tools and applications. If used individually, we suggest first reading through Chapters 1–4 in order to get a good understanding of the prerequisite knowledge in this book. Chapters 1, 2, and 3 will familiarize you with the concepts and vocabulary necessary to understand the future chapters. Chapter 4 introduces you to the companion toolkit MeTA, which is used in exercises in each chapter. We hope the exercises and chapter descriptions provide inspiration to work on your own text mining project. The provided code in MeTA should give a large head start and allow you to focus more on your contribution.

If used in class, there are several logical flows that an instructor may choose to take. As prerequisite knowledge, we assume some basic knowledge in probability and statistics as well as programming in a language such as C++ or Java. MeTA is written in modern C++, although some exercises may be accomplished only by modifying config files.

Instructors. We have gathered a logical and cohesive collection of topics that may be combined together for various course curricula. For example, Part 1 and Part 2 of the book may be used as an undergraduate introduction to *Information Retrieval* with a focus on how search engines work. Exercises assume basic programming experience and a little mathematical background in probability and statistics. A different undergraduate course may choose to survey

the entire book as an *Introduction to Text Data Mining*, while skipping some chapters in Part 2 that are more specific to search engine implementation and applications specific to the Web. Another choice would be using all parts as a supplemental graduate textbook, where there is still some emphasis on practical programming knowledge that can be combined with reading referenced papers in each chapter. Exercises for graduate students could be implementing some methods they read in the references into MeTA.

The exercises at the end of each chapter give students experience working with a powerful—yet easily understandable—text retrieval and mining toolkit in addition to written questions. In a programming-focused class, using the MeTA exercises is strongly encouraged. Programming assignments can be created from selecting a subset of exercises in each chapter. Due to the modular nature of the toolkit, additional programming experiments may be created by extending the existing system or implementing other well-known algorithms that do not come with MeTA by default. Finally, students may use components of MeTA they learned through the exercises to complete a larger final programming project. Using different corpora with the toolkit can yield different project challenges, e.g., review summary vs. sentiment analysis.

Practitioners. Most readers in industry would most likely use this book as a reference, although we also hope that it may serve as some inspiration in your own work. As with the student user suggestion, we think you would get the most of this book by first reading the initial three chapters. Then, you may choose a chapter relevant to your current interests and delve deeper or refresh your knowledge.

Since many applications in MeTA can be used simply via config files, we anticipate it as a quick way to get a handle on your dataset and provide some baseline results without any programming required.

The exercises at the end of each chapter can be thought of as default implementations for a particular task at hand. You may choose to include MeTA in your work since it uses a permissive free software license. In fact, it is dual-licensed under MIT and University of Illinois/NCSA licenses. Of course, we still encourage and invite you to share any modifications, extensions, and improvements with MeTA that are not proprietary for the benefit of all the readers.

No matter what your goal, we hope that you find this book useful and educational. We also appreciate your comments and suggestions for improvement of the book. Thanks for reading!

Bibliographic Notes and Further Reading

There are already multiple excellent text books in information retrieval (IR). Due to the long history of research in information retrieval and the fact that much foudational work has been done in 1960s, even some very old books such as van Rijsbergen [1979] and Salton and McGill [1983] and Salton [1989] remain very useful today. Another useful early book is Frakes and Baeza-Yates [1992]. More recent ones include Grossman and Frieder [2004], Witten et al. [1999], and Belew [2008]. The most recent ones are Manning et al. [2008], Croft et al. [2009], Büttcher et al. [2010], and Baeza-Yates and Ribeiro-Neto [2011]. Compared with these books, this book has a broader view of the topic of information retrieval and attempts to cover both text retrieval and text mining. While some existing books on IR have also touched some topics such as text categorization and text clustering, which we classify as text mining topics, no previous book has included an in-depth discussion of topic mining and analysis, an important family of techniques very useful for text mining. Recommender systems also seem to be missing in the existing books on IR, which we include as an alternative way to support users for text access complementary with search engines. More importantly, this book treats all these topics in a more systematic way than existing books by framing them in a unified coherent conceptual framework for managing and analyzing big text data; the book also attempts to minimize the gap between abstract explanation of algorithms and practical applications by providing a companion toolkit for many exercises. Readers who want to know more about the history of IR research and the major early milestones should take a look at the collection of readings in Sparck Jones and Willett [1997].

The topic of text mining has also been covered in multiple books (e.g., Feldman and Sanger [2007]). A major difference between this book and those is our emphasis on the integration of text mining and information retrieval with a belief that any text data application system must involve humans in the loop and search engines are essential components of any text mining systems to support two essential functions: (1) help convert a large raw text data set into a much smaller, but more relevant text data set which can be efficiently anlayzed by using a text mining algorithm (i.e., data reduction) and (2) help users verify the source text articles from which knowledge is discovered by a text mining algorithm (i.e., knowledge provenance). As a result, this book provides a more complete coverage of techniques required for developing big text data applications.

The focus of this book is on covering algorithms that are general and robust, which can be readily applied to any text data in any natural language, often with no or minimum human effort. An evitable cost of this focus is its lack of coverage

of some key techniques important for text mining, notably the information extraction (IE) techniques which are essential for text mining. We decided not to cover IE because the IE techniques tend to be language-specific and require non-trivial manual work by humans. Another reason is that many IE techniques rely on supervised machine learning approaches, which are well covered in many existing machine learning books (see, e.g., Bishop 2006, Mitchell 1997). Readers who are interested in knowing more about IE can start with the survey book [Sarawagi 2008] and review articles [Jiang 2012].

From an application perspective, another important topic missing in this book is information visualization, which is due to our focus on the coverage of models and algorithms. However, since every application system must have a user-friendly interface to allow users to optimally interact with a system, those readers who are interested in developing text data application systems will surely find it useful to learn more about user interface design. An excellent reference to start with is Hearst [2009], which also has a detailed coverage of information visualization.

Finally, due to our emphasis on breadth, the book does not cover any component algorithm in depth. To know more about some of the topics, readers can further read books in natural language processing (e.g., Jurafsky and Martin 2009, Manning and Schütze 1999), advanced books on IR (e.g., Baeza-Yates and Ribeiro-Neto [2011]), and books on machine learning (e.g., Bishop [2006]). You may find more specific recommendations of readings relevant to a particular topic in the Bibliographic Notes at the end of each chapter that covers the corresponding topic.

2 Background

This chapter contains background information that is necessary to know in order to get the most out of the rest of this book; readers who are already familiar with these basic concepts may safely skip the entire chapter or some of the sections. We first focus on some basic probability and statistics concepts required for most algorithms and models in this book. Next, we continue our mathematical background with an overview of some concepts in information theory that are often used in many text mining applications. The last section introduces the basic idea and problem setup of machine learning, particularly supervised machine learning, which is useful for *classification*, *categorization*, or *text-based prediction* in the text domain. In general, machine learning is very useful for many information retrieval and data mining tasks.

2.1 Basics of Probability and Statistics

As we will see later in this chapter and in many other chapters, probabilistic or statistical models play a very important role in text mining algorithms. This section gives every reader a sufficient background and vocabulary to understand these probabilistic and statistical approaches covered in the later chapters of the book.

A probability distribution is a way to assign likelihood to an event in some probability space Ω. As an example, let our probability space be a six-sided die. Each side has a different color. Thus, $\Omega = \{red, orange, yellow, green, blue, purple\}$ and an event is the act of rolling the die and observing a color.

We can quantify the uncertainty of rolling the die by declaring a **probability distribution** over all possible events. Assuming we have a fair die, the probability of rolling any specific color is $\frac{1}{6}$, or about 16%. We can represent our probability distribution as a collection of probabilities such as

$$\theta = \left\{\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}\right\},$$

where the first index corresponds to $p(\text{red}) = \frac{1}{6}$, the second index corresponds to $p(\text{orange}) = \frac{1}{6}$, and so on. But what if we had an unfair die? We could use a different probability distribution θ' to model events concerning it:

$$\theta' = \left\{ \frac{1}{3}, \frac{1}{3}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12} \right\}.$$

In this case, red and orange are assumed to be rolled more often than the other colors. Be careful to note the difference between the sample space Ω and the defined probability model θ used to quantify its uncertainty. In our text mining tasks, we usually try to estimate θ given some knowledge about Ω. The different methods to estimate θ will determine how accurate or useful the probabilistic model is.

Consider the following notation:

$$x \sim \theta.$$

We read this as *x is drawn from theta*, or the **random variable** x is drawn from the probability distribution θ. The random variable x takes on each value from Ω with a certain probability defined by θ. For example, if we had $x \sim \theta'$, then there is a $\frac{2}{3}$ chance that x is either red or orange.

In our text application tasks, we usually have Ω as V, the vocabulary of some text corpus. For example, the vocabulary could be

$$V = \{a, and, apple, \ldots, zap, zirconium, zoo\}$$

and we could model the text data with a probability distribution θ. Thus, if we have some word w we can write $p(w \mid \theta)$ (read as *the probability of w given* θ). If w is the word *data*, we might have $p(w = \text{data} \mid \theta) = 0.003$ or equivalently $p_\theta(w = \text{data}) = 0.003$.

In our examples, we have only considered **discrete probability distributions**. That is, our models only assign probabilities for a finite (discrete) set of outcomes. In reality, there are also **continuous probability distributions**, where there are an infinite number of "events" that are not countable. For example, the normal (or Gaussian) distribution is a continuous probability distribution that assigns real-valued probabilities according to some parameters. We will discuss continuous distributions as necessary in later chapters. For now, though, it's sufficient to understand that we can use a discrete probability distribution to model the probability of observing a single word in a vocabulary V.

The Kolmogorov axioms describe facts about probability distributions in general (both discrete and continuous). We discuss them now, since they are a good sanity check when designing your own models. A valid probability distribution θ with probability space Ω must satisfy the following three axioms.

- Each event has a probability between zero and one:

$$0 \leq p_\theta(\omega \in \Omega) \leq 1. \tag{2.1}$$

- An event not in Ω has probability zero, and the probability of any event occurring from Ω is one:

$$p_\theta(\omega') = 0, \omega' \notin \Omega \quad \text{and} \quad p_\theta(\Omega) = 1. \tag{2.2}$$

- The probability of all (disjoint) events sums to one:

$$\sum_{\omega \in \Omega} p_\theta(\omega) = 1. \tag{2.3}$$

Note that, strictly speaking, an event is defined as a subset of the probability space Ω, and we say that an event happens if and only if the outcome from a random experiment (i.e., randomly drawing an outcome from Ω) is in the corresponding subset of outcomes defined by the event. Thus, it is easy to understand that the special event corresponding to the empty subset is an impossible event with a probability of zero, whereas the special event corresponding to the complete set Ω itself always happens and so has a probability of 1.0. As a special case, we can consider an event space with only those events that each have precisely one element of outcome, which is exactly what we assumed when talking about a distribution over all the words. Here each word corresponds to the event defined by the subset with the word as the only element; clearly, such an event happens if and only if the outcome is the corresponding word.

2.1.1 Joint and Conditional Probabilities

For this section, let's modify our original die rolling example. We will keep the original distribution as θ_C, indicating the color probabilities:

$$\theta_C = \left\{ \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6} \right\}.$$

Let's also assume that each color is represented by a particular shape. This makes our die look like

where the colors of the shapes are, red, orange, yellow, blue, green, and purple, respectively.

We can now create another distribution for the shape θ_S. Let each index in θ_S represent $p(\text{square})$, $p(\text{circle})$, $p(\text{triangle})$, respectively. That gives

$$\theta_S = \left\{\frac{1}{3}, \frac{1}{2}, \frac{1}{6}\right\}.$$

Then we can let $x_C \sim \theta_C$ represent the color random variable and let $x_S \sim \theta_S$ represent the shape random variable. We now have two variables to work with.

A **joint probability** measures the likelihood that two events occur simultaneously. For example, what is the probability that $x_C = \text{red}$ and $x_S = \text{circle}$? Since there are no red circles, this has probability zero. How about $p(x_C = \text{green}, x_S = \text{circle})$? This notation signifies the joint probability of the two random variables. In this case, the joint probability is $\frac{1}{6}$ because there is only one green circle.

Consider a modified die:

where we changed the color of the blue circle (the fourth element in the set) to green. Thus, we now have two green circles instead of one green and one blue. What would $p(x_C = \text{green}, x_S = \text{circle})$ be? Since two out of the six elements satisfy both these criteria, the answer is $\frac{2}{6} = \frac{1}{3}$. As another example, if we had a 12-sided fair die with 5 green circles and 7 other combinations of shape and color, then $p(x_C = \text{green}, x_S = \text{circle}) = \frac{5}{12}$.

A **conditional probability** measures the likelihood that one event occurs given that another event has already occurred. Let's use the original die with six unique colors. Say we know that a square was rolled. With that information, what is the probability that the color is red? How about purple? We can write this as $p(x_C = \text{red} \mid x_S = \text{square})$. Since we know there are two squares, of which one is red, $p(\text{red} \mid \text{square}) = \frac{1}{2}$.

We can write the conditional probabilities for two random variables X and Y based on their joint probability with the following equation:

$$p(X = x \mid Y = y) = \frac{p(X = x, Y = y)}{p(Y = y)}. \tag{2.4}$$

The numerator $p(X = x, Y = y)$ is the probability of exactly the configuration we're looking for (i.e., both x and y have been observed), which is normalized by $p(Y = y)$, the probability that the condition is true (i.e., y has been observed). Using this knowledge, we can calculate $p(x_C = \text{green} \mid x_S = \text{circle})$:

$$p(x_C = \text{green} \mid x_S = \text{circle}) = \frac{p(x_C = \text{green}, x_S = \text{circle})}{p(x_S = \text{circle})} = \frac{1/6}{1/2} = \frac{1}{3}.$$

One other important concept to mention is **independence**. In the previous examples, the two random variables were **dependent**, meaning the value of one will influence the value of the other. Consider another situation where we have $c_1, c_2 \sim \theta_C$. That is, we draw two colors from the color distribution. Does the knowledge of c_1 inform the probability of c_2? No, since each draw is done "independently" of the other. In the case where two random variables X and Y are independent, $p(X, Y) = p(X)p(Y)$. Can you see why this is the case?

Both conditional and joint probabilities can be used to answer interesting questions about text. For example, given a document, what is the probability of observing the word *information* and *retrieval* in the same sentence? What is the probability of observing *retrieval* if we know *information* has occurred?

2.1.2 Bayes' Rule

Bayes' rule may be derived using the definition of conditional probability:

$$p(X \mid Y) = \frac{p(X, Y)}{p(Y)} \quad \text{and} \quad p(Y \mid X) = \frac{p(Y, X)}{p(X)}.$$

Therefore, setting the two joint probabilities equal,

$$p(X \mid Y)p(Y) = p(X, Y) = p(Y \mid X)p(X).$$

We can simplify them as

$$p(X \mid Y) = \frac{p(Y \mid X)p(X)}{p(Y)}. \tag{2.5}$$

The above formula is known as Bayes' rule, named after the Reverend Thomas Bayes (1701–1761). This rule has widespread applications. In this book, you will see heavy use of this formula in the text categorization chapter as well as the topic analysis chapter, among others. We will leave it up to the individual chapters to explain their use of this rule in their implementation. Essentially, though, Bayes' rule can be used to make inference about a hypothesis based on the observed evidence related to the hypothesis.

Specifically, we may view random variable X as denoting our hypothesis, and Y as denoting the observed evidence. $p(X)$ can thus be interpreted as our prior belief about which hypothesis is true; it is "prior" because it is our belief *before* we have any knowledge about evidence Y. In contrast, $p(X \mid Y)$ encodes our posterior belief about the hypothesis since it is our belief *after* knowing evidence Y. Bayes' rule is seen to connect the prior belief and posterior belief, and provide a way to update the prior belief $p(X)$ based on the likelihood of the observed evidence Y and obtain the posterior belief $p(X \mid Y)$. It is clear that if X and Y are independent, then no updating will happen as in this case, $p(X \mid Y) = p(X)$.

2.1.3 Coin Flips and the Binomial Distribution

In most discussions on probability, a good example to investigate is flipping a coin. For example, we may be interested in modeling the presence or absence of a particular word in a text document, which can be easily mapped to a coin flipping problem. There are two possible outcomes in coin flipping: heads or tails. The probability of heads is denoted as θ, which means the probability of tails is $1 - \theta$.

To model the probability of success (in our case, "heads"), we can use the Bernoulli distribution. The Bernoulli distribution gives the probability of success for a single event—flipping the coin once. If we want to model n throws and find the probability of k successes, we instead use the binomial distribution. The binomial distribution is a discrete distribution since k is an integer. We can write it as

$$p(k \text{ heads}) = \binom{n}{k} \theta^k (1-\theta)^{n-k}. \tag{2.6}$$

We can also write it as follows:

$$p(k \text{ heads}) = \frac{n!}{k!(n-k)!} \theta^k (1-\theta)^{n-k}. \tag{2.7}$$

But why is it this formula? Well, let's break it apart. If we have n total binary trials, and want to see k heads, that means we must have flipped k heads and $n - k$ tails. The probability of observing each of the k heads is θ, while the probability of observing each of the remaining $n - k$ tails is $1 - \theta$. Since we assume all these flips are independent, we simply multiply all the outcomes together. Since we don't care about the order of the outcomes, we additionally multiply by the number of possible ways to choose k items from a set of n items.

What if we do care about the order of the outcomes? For example, what is the probability of observing the particular sequence of outcomes (h, t, h, h, t) where h and t denote heads and tails, respectively? Well, it is easy to see that the probability

of observing this sequence is simply the product of observing each event, i.e., $\theta \times (1-\theta) \times \theta \times \theta \times (1-\theta) = \theta^3(1-\theta)^2$ with no adjustment for different orders of observing three heads and two tails.

The more commonly used multinomial distribution in text analysis, which models the probability of seeing a word in a particular scenario (e.g., in a document), is very similar to this Bernoulli distribution, just with more than two outcomes.

2.1.4 Maximum Likelihood Parameter Estimation

Now that we have a model for our coin flipping, how can we estimate its parameters given some observed data? For example, maybe we observe the data D that we discussed above where $n = 5$:

$$D = \{h, t, h, h, t\}.$$

Now we would like to figure out what θ is based on the observed data. Using maximum likelihood estimation (MLE), we choose the θ that has the highest likelihood given our data, i.e., choose the θ such that the probability of observed data is maximized.

To compute the MLE, we would first write down the likelihood function, i.e., $p(D \mid \theta)$, which is $\theta^3(1-\theta)^2$ as we explained earlier. The problem is thus reduced to find the θ that maximizes the function $f(\theta) = \theta^3(1-\theta)^2$. Equivalently, we can attempt to maximize the log-likelihood: $\log f(\theta) = 3\log\theta + 2\log(1-\theta)$, since logarithm transformation preserves the order of values. Using knowledge of calculus, we know that a necessary condition for a function to achieve a maximum value at a θ value is that the derivative at the same θ value is zero. Thus, we just need to solve the following equation:

$$\frac{d \log f(\theta)}{d\theta} = \frac{3}{\theta} - \frac{2}{1-\theta} = 0,$$

and we easily find that the solution is $\theta = 3/5$.

More generally, let H be the number of heads and T be the number of tails. The MLE of the probability of heads is given by:

$$\begin{aligned}\theta_{MLE} &= \arg\max_{\theta} p(D \mid \theta) \\ &= \arg\max_{\theta} \theta^H (1-\theta)^T \\ &= \frac{H}{H+T}.\end{aligned}$$

The notation arg max represents the argument (i.e., θ in this case) that makes the likelihood function (i.e., $p(D \mid \theta)$) reach its maximum. Thus, the value of an arg max expression stays the same if we perform any monotonic transformation of the function inside arg max. This is why we could use the logarithm transformation in the example above, which made it easier to compute the derivative.

The solution to MLE shown above should be intuitive: the θ that maximizes our data likelihood is just the ratio of heads. It is a general characteristic of the MLE that the estimated probability is the normalized counts of the corresponding events denoted by the probability. As an example, the MLE of a multinomial distribution (which will be further discussed in detail later in the book) gives each possible outcome a probability proportional to the observed counts of the outcome. Note that a consequence of this is that all unobserved outcomes would have a zero probability according to MLE. This is often not reasonable especially when the data sample is small, a problem that motivates Bayesian parameter estimation which we discuss below.

2.1.5 Bayesian Parameter Estimation

One potential problem of MLE is that it is often inaccurate when the size of the data sample is small since it always attempts to fit the data as well as possible. Consider an extreme example of observing just two data points of flipping a coin which happen to be all heads. The MLE would say that the probability of heads is 1.0 while the probability of tails is 0. Such an estimate is intuitively inaccurate even though it maximizes the probability of the observed two data points.

This problem of "overfitting" can be addressed and alleviated by considering the uncertainty on the parameter and using Bayesian parameter estimation instead of MLE. In Bayesian parameter estimation, we consider a distribution over all the possible values for the parameter; that is, we treat the parameter itself as a random variable.

Specifically, we may use $p(\theta)$ to represent a distribution over all possible values for θ, which encodes our prior belief about what value is the true value of θ, while the data D provide evidence for or against that belief. The prior belief $p(\theta)$ can then be updated based on the observed evidence. We'll use Bayes' rule to rewrite $p(\theta \mid D)$, or our belief of the parameters given data, as

$$p(\theta \mid D) = \frac{p(D \mid \theta)p(\theta)}{p(D)}, \tag{2.8}$$

where $p(D)$ can be calculated by summing over all configurations of θ. For a continuous distribution, that would be

$$p(D) = \int_{\theta'} p(\theta')p(D \mid \theta')d\theta' \tag{2.9}$$

which means the probability for a particular θ is

$$p(\theta \mid D) = \frac{p(D \mid \theta)p(\theta)}{\int_{\theta'} p(\theta')p(D \mid \theta')d\theta'}. \tag{2.10}$$

We have special names for these quantities:

- $p(\theta \mid D)$: the posterior probability of θ
- $p(\theta)$: the prior probability of θ
- $p(D \mid \theta)$: the likelihood of D
- $\int_{\theta'} p(\theta')p(D \mid \theta')d\theta'$: the marginal likelihood of D

The last one is called the marginal likelihood because the integration "marginalizes out" (removes) the parameter θ from the equation. Since the likelihood of the data remains constant, observing the constraint that $p(\theta \mid D)$ must sum to one over all possible values of θ, we usually just say

$$p(\theta \mid D) \propto p(\theta)p(D \mid \theta).$$

That is, the posterior is proportional to the prior times the likelihood.

The posterior distribution of the parameter θ fully characterizes the uncertainty of the parameter value and can be used to infer any quantity that depends on the parameter value, including computing a point estimate of the parameter (i.e., a single value of the parameter). There are multiple ways to compute a point estimate based on a posterior distribution. One possibility is to compute the mean of the posterior distribution, which is given by the weighted sum of probabilities and the parameter values. For a discrete distribution,

$$E[X] = \sum_x xp(x) \tag{2.11}$$

while in a continuous distribution,

$$E[X] = \int_x xf(x)dx \tag{2.12}$$

Sometimes, we are interested in using the mode of the posterior distribution as our estimate of the parameter, which is called Maximum a Posteriori (MAP) estimate, given by:

$$\theta_{MAP} = \arg\max_\theta p(\theta \mid D) = \arg\max_\theta p(D \mid \theta)p(\theta). \tag{2.13}$$

Here it is easy to see that the MAP estimate would deviate from the MLE with consideration of maximizing the probability of the parameter according to our prior belief encoded as $p(\theta)$. It is through the use of appropriate prior that we can address the overfitting problem of MLE since our prior can strongly prefer an estimate where neither heads, nor tails should have a zero probability.

For a continuation and more in-depth discussion of this material, consult Appendix A.

2.1.6 Probabilistic Models and Their Applications

With the statistical foundation from the previous sections, we can now start to see how we might apply a probabilistic model to text analysis.

In general, in text processing, we would be interested in a probabilistic model for text data, which defines distributions over sequences of words. Such a model is often called statistical language model, or a generative model for text data (i.e., a probabilistic model that can be used for sampling sequences of words).

As we started to explain previously, we usually treat the sample space Ω as V, the set of all observed words in our corpus. That is, we define probability distributions over words from our dataset, which are essentially multinomial distributions if we do not consider the order of words. While there are many more sophisticated models for text data (see, e.g., Jelinek [1997]), this simplest model (often called unigram language model) is already very useful for a number of tasks in text data management and analysis due to the fact that the words in our vocabulary are very well designed meaningful basic units for human communications.

For now, we can discuss the general framework in which statistical models are "learned." Learning a model means estimating its parameters. In the case of a distribution over words, we have one parameter for each element in V. The workflow looks like the following.

1. Define the model.
2. Learn its parameters.
3. Apply the model.

The first step has already been addressed. In our example, we wish to capture the probabilities of individual words occurring in our corpus. In the second step, we need to figure out actually how to set the probabilities for each word. One obvious way would be to calculate the probability of each individual word in the corpus itself. That is, the count of a unique word w_i divided by the total number of words

in the corpus could be the value of $p(w_i \mid \theta)$. This can be shown to be the solution of the MLE of the model. More sophisticated models and their parameter estimation will be discussed later in the book. Finally, once we have θ defined, what can we actually do with it? One use case would be analyzing the probability of a specific subset of words in the corpus, and another could be observing unseen data and calculating the probability of seeing the words in the new text. It is often possible to design the model such that the model parameters would encode the knowledge we hope to discover from text data. In such a case, the estimated model parameters can be directly used as the output (result) of text mining.

Please keep in mind that probabilistic models are a general tool and don't only have to be used for text analysis—that's just our main application!

2.2 Information Theory

Information theory deals with uncertainty and the transfer or storage of quantified information in the form of bits. It is applied in many fields, such as electrical engineering, computer science, mathematics, physics, and linguistics. A few concepts from information theory are very useful in text data management and analysis, which we introduce here briefly. The most important concept of information theory is **entropy**, which is a building block for many other measures.

The problem can be formally defined as the quantified uncertainty in predicting the value of a random variable. In the common example of a coin, the two values would be 1 or 0 (depicting heads or tails) and the random variable representing these outcomes is X. In other words,

$$X = \begin{cases} 1 & \text{if heads} \\ 0 & \text{if tails.} \end{cases}$$

The more random this random variable is, the more difficult the prediction of heads or tails will be. How does one quantitatively measure the randomness of a random variable like X? This is precisely what entropy does.

Roughly, the entropy of a random variable X, $H(X)$, is a measure of expected number of bits needed to represent the outcome of an event $x \sim X$. If the outcome is known (completely certain), we don't need to represent any information and $H(X) = 0$. If the outcome is unknown, we would like to represent the outcome in bits as efficiently as possible. That means using fewer bits for common occurrences and more bits when the event is less likely. Entropy gives us the expected number

of bits for any $x \sim X$ using the formula

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x). \tag{2.14}$$

In the cases where we have $\log_2 0$, we generally just define this to be 0 since $\log_2 0$ is undefined. We will get different $H(X)$ for different random variables X.

The exact theory and reasoning behind this formula are beyond the scope of this book, but it suffices to say that $H(X) = 0$ means there is no randomness, $H(X) = 1$ means there is complete randomness in that all events are equally likely. Thus, the amount of randomness varies from 0 to 1. For our coin example where the sample space is two events (heads or tails), the entropy function looks like

$$H(X) = -p(X = 0) \log_2 p(X = 0) - p(X = 1) \log_2 p(X = 1).$$

For a fair coin, we would have $p(X = 1) = p(X = 0) = \frac{1}{2}$. To calculate $H(X)$, we'd have the calculation

$$H(X) = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1,$$

whereas for a completely biased coin with $p(X = 1) = 1$, $p(X = 0) = 0$ we would have

$$H(X) = -0 \log_2 0 - 1 \log_2 1 = 0.$$

For this example, we had only two possible outcomes (i.e., a binary random variable). As we can see from the formula, this idea of entropy easily generalizes to random variables with more than two outcomes; in those cases, the sum is over more than two elements.

If we plot $H(X)$ for our coin example against the probability of heads $p(X = 1)$, we receive a plot like the one shown in Figure 2.1. At the two ends of the x-axis, the probability of $X = 1$ is either very small or very large. In both these cases, the entropy function has a low value because the outcome is not very random. The most random is when $p(X = 1) = \frac{1}{2}$. In that case, $H(X) = 1$, the maximum value. Since the two probabilities are symmetric, we get a symmetric inverted U-shape as the plot of $H(X)$ as $p(X = 1)$ varies.

It's a good exercise to consider when a particular random variable (not just the coin example) has a maximum or minimal value. In particular, let's think about some special cases. For example, we might have a random variable Y that always takes a value of 1. Or, there's a random variable Z that is equally likely to take a value of 1, 2, or 3. In these cases, $H(Y) < H(Z)$ since the outcome of Y is much

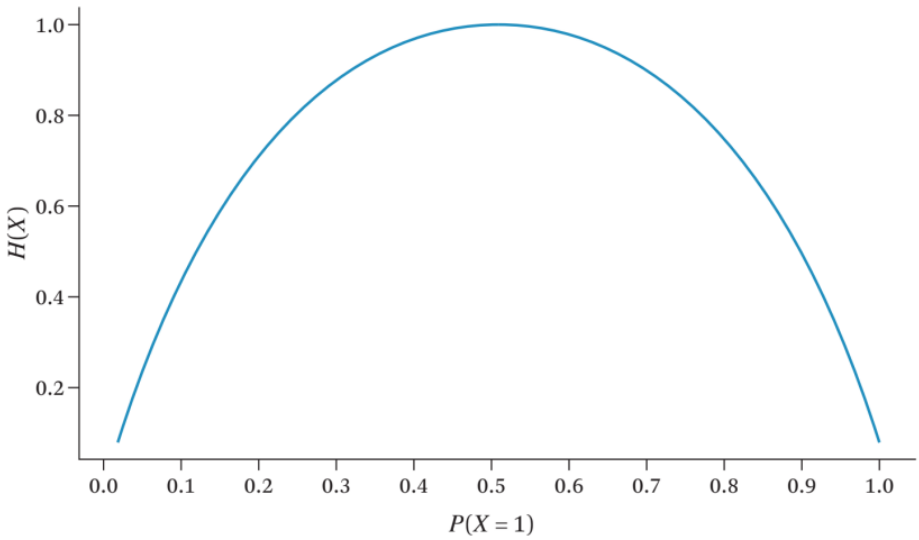

Figure 2.1 **Entropy as a measure of randomness of a random variable.**

easier to predict than the outcome of Z. This is precisely what entropy captures. You can calculate $H(Y)$ and $H(Z)$ to confirm this answer.

For our applications, it may be useful to consider the entropy of a word w in some context. Here, high-entropy words would be harder to predict. Let W be the random variable that denotes whether a word occurs in a document in our corpus. Say $W = 1$ if the word occurs and $W = 0$ otherwise. How do you think $H(W_{the})$ compares to $H(W_{computer})$? The entropy of the word *the* is close to zero since it occurs everywhere. It's not surprising to see this word in a document, thus it is easy to predict that $W_{the} = 1$. This case is just like the biased coin that always lands one way. The word *computer*, on the other hand, is a less common word and is harder to predict whether it occurs or not, so the entropy will be higher.

When we attempt to quantify uncertainties of conditional probabilities, we can also define conditional entropy $H(X \mid Y)$, which indicates the expected uncertainty of X given that we observe Y, where the expectation is taken under the distribution of all possible values of Y. Intuitively, if X is completely determined by Y, then $H(X \mid Y) = 0$ since once we know Y, there would be no uncertainty in X, whereas if X and Y are independent, then $H(X \mid Y)$ would be the same as the original entropy of X, i.e., $H(X \mid Y) = H(X)$ since knowing Y does not help at all in resolving the uncertainty of X.

Another useful concept is mutual information defined on two random variables, $I(X;Y)$, which is defined as the reduction of entropy of X due to knowledge about another random variable Y, i.e.,

$$I(X;Y) = H(X) - H(X \mid Y). \tag{2.15}$$

It can be shown that mutual information can be equivalently written as

$$I(X;Y) = H(Y) - H(Y \mid X). \tag{2.16}$$

It is easy to see that $I(X;Y)$ tends to be large if X and Y are correlated, whereas $I(X;Y)$ would be small if X and Y are not so related; indeed, in the extreme case when X and Y are completely independent, there would be no reduction of entropy, and thus $H(X) = H(X \mid Y)$, and $I(X;Y) = 0$. However, if X is completely determined by Y, then $H(X \mid Y) = 0$, thus $I(X;Y) = H(X)$. Intuitively, mutual information can measure the correlation of two random variables. Clearly as a correlation measure on X and Y, mutual information is symmetric.

Applications of these basic concepts, including entropy, conditional entropy, and mutual information will be further discussed later in this book.

2.3 Machine Learning

Machine learning is a very important technique for solving many problems, and has very broad applications. In text data management and analysis, it has also many uses. Any in-depth treatment of this topic would clearly be beyond the scope of this book, but here we introduce some basic concepts in machine learning that are needed to better understand the content later in the book.

Machine learning techniques can often be classified into two types: **supervised learning** and **unsupervised learning**. In supervised learning, a computer would learn how to compute a function $\hat{y} = f(x)$ based on a set of examples of the input value x (called training data) and the corresponding expected output value y. It is called "supervised" because typically the y values must be provided by humans for each x, and thus the computer receives a form of supervision from the humans. Once the function is learned, the computer would be able to take unseen values of x and compute the function $f(x)$.

When y takes a value from a finite set of values, which can be called labels, a function $f(\cdot)$ can serve as a classifier in that it can be used to map an instance x to the "right" label (or multiple correct labels when appropriate). Thus, the problem can be called a classification problem. The simplest case of the classification problem is when we have just two labels (known as binary classification). When y

takes a real value, the problem is often called a regression problem. Both forms of the problem can also be called prediction when our goal is mainly to infer the unknown y for a given x; the term "prediction" is especially meaningful when y is some property of a future event.

In text-based applications, both forms may occur, although the classification problem is far more common, in which case the problem is also called *text categorization* or *text classification*. We dedicate a chapter to this topic later in the book (Chapter 15). The regression problem may occur when we use text data to predict another non-text variable such as sentiment rating or stock prices; both cases are also discussed later.

In classification as well as regression, the (input) data instance x is often represented as a feature vector where each feature provides a potential clue about which y value is most likely the value of $f(x)$. What the computer learns from the training data is an optimal way to combine these features with weights on them to indicate their importance and their influence on the final function value y. "Optimal" here simply means that the prediction error on the training data is minimum, i.e., the predicted $\hat{y}$ values are maximally consistent with the true y values in the training data.

More formally, let our collection of objects be $\mathbf{X}$ such that $x_i \in \mathbf{X}$ is a feature vector that represents object i. A feature is an attribute of an object that describes it in some way. For example, if the objects are news articles, one feature could be whether the word *good* occurred in the article. All these different features are part of a document's feature vector, which is used to represent the document. In our cases, the feature vector will usually have to do with the words that appear in the document.

We also have $\mathbf{Y}$, which is the set of possible labels for each object. Thus, y_i may be *sports* in our news article classification setup and y_j could be *politics*.

A classifier is a function $f(\cdot)$ that takes a feature vector as input and outputs a predicted label $\hat{y} \in \mathbf{Y}$. Thus, we could have $f(x_i) = \textit{sports}$, meaning $\hat{y} = \textit{sports}$. If the true y is also *sports*, the classifier was correct in its prediction.

Notice how we can only evaluate a classification algorithm if we know the true labels of the data. In fact, we will have to use the true labels in order to learn a good function $f(\cdot)$ to take unseen feature vectors and classify them. For this reason, when studying machine learning algorithms, we often split our corpus $\mathbf{X}$ into two parts: **training data** and **testing data**. The training portion is used to build the classifier, and the testing portion is used to evaluate the performance (e.g., determine how many correct labels were predicted). In applications, the training data are generally

all the labelled examples that we can generate, and the test cases are the data points, to which we would like to apply our machine learning program.

But what does the function $f(\cdot)$ actually do? Consider a very simple example that determines whether a news article has positive or negative sentiment, i.e., $\mathbf{Y} = \{positive, negative\}$:

$$f(x) = \begin{cases} positive & \text{if } x\text{'s count for the term } good \text{ is greater than 1} \\ negative & \text{otherwise.} \end{cases}$$

Of course, this example is overly simplified, but it does demonstrate the basic idea of a classifier: it takes a feature vector as input and outputs a class label. Based on the training data, the classifier may have determined that positive sentiment articles contain the term *good* more than once; therefore, this knowledge is encoded in the function. In Chapter 15, we will investigate some specific algorithms for creating the function $f(\cdot)$ based on the training data. Other topics such as feedback for information retrieval (Chapter 7) and sentiment analysis (Chapter 18) make use of classifiers, or resemble them. For this reason, it's good to know what machine learning is and what kinds of problems it can solve.

In contrast to supervised learning, in unsupervised learning we only have the data instances $\mathbf{X}$ without knowing $\mathbf{Y}$. In such a case, obviously we cannot really know how to compute y based on an x. However, we may still learn latent properties or structures of $\mathbf{X}$. Since there is no human effort involved, such an approach is called unsupervised. For example, the computer can learn that some data instances are very similar, and the whole dataset can be represented by three major clusters of data instances such that in each cluster, the data instances are all very similar. This is essentially the clustering technique that we will discuss in Chapter 14. Another form of unsupervised learning is to design probabilistic models to model the data (called "generative models") where we can embed interesting parameters that denote knowledge that we would like to discover from the data. By fitting the model to our data, we can estimate the parameter values that can best explain the data, and treat the obtained parameter values as the knowledge discovered from the data. Applications of such an approach in analyzing latent topics in text are discussed in detail in Chapter 17.

Bibliographic Notes and Further Reading

Detailed discussion of the basic concepts in probability and statistics can be found in many textbooks such as Hodges and Lehmann [1970]. An excellent introduction to the maximum likelihood estimation can be found in Myung [2003]. An accessi-

ble comprehensive introduction to Bayesian statistics is given in the book *Bayesian Data Analysis* [Gelman et al. 1995]. Cover and Thomas [1991] provide a comprehensive introduction to information theory. There are many books on machine learning where a more rigorous introduction to the basic concepts in machine learning as well as many specific machine learning approaches can be found (e.g., Bishop 2006, Mitchell 1997).

Exercises

2.1. What can you say about $p(X \mid Y)$ if we know X and Y are independent random variables? Prove it.

2.2. In an Information Retrieval course, there are 78 computer science majors, 21 electrical and computer engineering majors, and 10 library and information science majors. Two students are randomly selected from the course. What is the probability that they are from the same department? What is the probability that they are from different departments?

2.3. Use Bayes' rule to solve the following problem. One third of the time, Milo takes the bus to work and the other times he takes the train. The bus is less reliable, so he gets to work on time only 50% of the time. If taking the train, he is on time 90% of the time. Given that he was on time on a particular day, what is the probability that Milo took the bus?

2.4. In a game based on a deck of 52 cards, a single card is drawn. Depending on the type of card, a certain value is either won or lost. If the card is one of the four aces, \$10 is won. If the card is one of the four kings, \$5 is won. If the card is one of the eleven diamonds that is not a king or ace, \$2 is won. Otherwise, \$1 is lost. What are the expected winnings or losings after drawing a single card? (Would you play?)

2.5. Consider the game outlined in the previous question. Imagine that two aces were drawn, leaving 50 cards remaining. What is the expected value of the next draw?

2.6. In the information theory section, we defined three random variables X, Y, and Z when discussing entropy. We compared $H(Y)$ with $H(Z)$. How does $H(X)$ compare to the other two entropies?

2.7. In the information theory section, we compared the entropy of the word *the* to that of the word *unicorn*. In general, what types of words have a high entropy and what types of words have a low entropy? As an example, consider a corpus of ten

documents where *the* occurs in all documents, *unicorn* appears in five documents, and *Mercury* appears in one document. What would be the entropy value of each?

2.8. Brainstorm some different features that may be good for the sentiment classification task outlined in this chapter. What are the strengths and weaknesses of such features?

2.9. Consider the following scenario. You are writing facial recognition software that determines whether there is a face in a given image. You have a collection of 100, 000 images with the correct answer and need to determine if there are faces in new, unseen images.

Answer the following questions.

(a) Is this supervised learning or unsupervised learning?

(b) What are the labels or values we are predicting?

(c) Is this binary classification or multiclass classification? (Or neither?)

(d) Is this a regression problem?

(e) What are the features that could be used?

2.10. Consider the following scenario. You are writing essay grading software that takes in a student essay and produces a score from 0–100%. To design this system, you are given essays from the past year which have been graded by humans. Your task is to use the system with the current year's essays as input.

Answer the same questions as in Exercise 2.9.

2.11. Consider the following scenario. You are writing a tool that determines whether a given web page is one of

- personal home page,
- links to a personal home page, or
- neither of the above.

To help you in your task, you are given 5, 000, 000 pages that are already labeled.

Answer the same questions as in Exercise 2.9.

3 Text Data Understanding

In this chapter, we introduce basic concepts in text data understanding through natural language processing (NLP). NLP is concerned with developing computational techniques to enable a computer to understand the meaning of natural language text. NLP is a foundation of text information systems because how effective a TIS is in helping users access and analyze text data is largely determined by how well the system can understand the content of text data. Content analysis is thus logically the first step in text data analysis and management.

While a human can instantly understand a sentence in their native language, it is quite challenging for a computer to make sense of one. In general, this may involve the following tasks.

Lexical analysis. The purpose of lexical analysis is to figure out what the basic meaningful units in a language are (e.g., words in English) and determine the meaning of each word. In English, it is rather easy to determine the boundaries of words since they are separated by spaces, but it is non-trivial to find word boundaries in some other languages such as Chinese where there is no clear delimiter to separate words.

Syntactic analysis. The purpose of syntactic analysis is to determine how words are related with each other in a sentence, thus revealing the syntactic structure of a sentence.

Semantic analysis. The purpose of semantic analysis is to determine the meaning of a sentence. This typically involves the computation of meaning of a whole sentence or a larger unit based on the meanings of words and their syntactic structure.

Pragmatic analysis. The purpose of pragmatic analysis is to determine meaning in context, e.g., to infer the speech acts of language. Natural language is used by humans to communicate with each other. A deeper understanding

of natural language than semantic analysis is thus to further understand the purpose in communication.

Discourse analysis. Discourse analysis is needed when a large chunk of text with multiple sentences is to be analyzed; in such a case, the connections between these sentences must be considered and the analysis of an individual sentence must be placed in the appropriate context involving other sentences.

In Figure 3.1, we show what is involved in understanding a very simple English sentence *"A dog is chasing a boy on the playground."* The lexical analysis in this case involves determining the syntactic categories (parts of speech) of all the words (for example, *dog* is a noun and *chasing* is a verb). Syntactic analysis is to determine that *a* and *boy* form a noun phrase. So do *the* and *playground*, and *on the playground* is a prepositional phrase. Semantic analysis is to map noun phrases to entities and verb phrases to relations so as to obtain a formal representation of the meaning of the sentence. For example, the noun phrase *a boy* can be mapped to a semantic entity denoting a boy (i.e., `b1`), and *a dog* to an entity denoting a dog (i.e., `d1`). The verb phrase can be mapped to a relation predicate `chasing(d1,b1,p1)` as shown in the figure. Note that with this level of understanding, one may also infer additional information based on any relevant common sense knowledge. For example, if we assume that if someone is being chased, he or she may be scared, we could infer that the boy being chased (`b1`) may be scared. Finally, pragmatic analysis might further reveal that the person who said this sentence might intend to request an action, such as reminding the owner of the dog to take the dog back.

While it is possible to derive a clear semantic representation for a simple sentence like the one shown in Figure 3.1, it is in general very challenging to do this kind of analysis for unrestricted natural language text. The main reason for this difficulty is because natural language is designed to make human communication efficient; this is in contrast with a programming language which is designed to facilitate computer understanding. Specifically, there are two reasons why NLP is very difficult. (1) We omit a lot of "common sense" knowledge in natural language communication because we assume the hearer or reader possesses such knowledge (thus there's no need to explicitly communicate it). (2) We keep a lot of ambiguities, which we assume the hearer/reader knows how to resolve (thus there's no need to waste words to clarify them). As a result, natural language text is full of ambiguity, and resolving ambiguity would generally involve reasoning with a large amount of common-sense knowledge, which is a general difficult challenge in artificial intel-

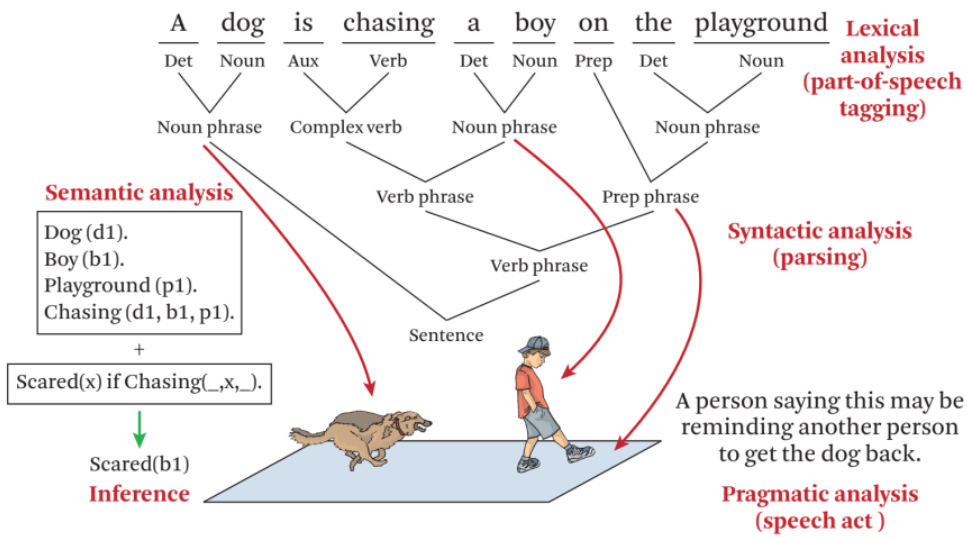

Figure 3.1 **An example of tasks in natural language understanding.**

ligence. In this sense, NLP is "AI complete", i.e., as difficult as any other difficult problems in artificial intelligence.

The following are a few examples of specific challenges in natural language understanding.

Word-level ambiguity. A word may have multiple syntactic categories and multiple senses. For example, *design* can be a noun or a verb (**ambiguous POS**); *root* has multiple meanings even as a noun (**ambiguous sense**).

Syntactic ambiguity. A phrase or a sentence may have multiple syntactic structures. For example, *natural language processing* can have two different interpretations: "processing of natural language" vs. "natural processing of language" (**ambiguous modification**). Another example: *A man saw a boy with a telescope* has two distinct syntactic structures, leading to a different result regarding who had the telescope (**ambiguous prepositional phrase (PP) attachment**).

Anaphora resolution. What exactly a pronoun refers to may be unclear. For example, in *John persuaded Bill to buy a TV for himself*, does *himself* refer to John or Bill?

Presupposition. *He has quit smoking* implies that he smoked before; making such inferences in a general way is difficult.

3.1 History and State of the Art in NLP

Research in NLP dated back to at least the 1950s when researchers were very optimistic about having computers that understood human language, particularly for the purpose of machine translation. Soon however, it was clear, as stated in Bar-Hillel's report in 1960, that fully-automatic high-quality translation could not be accomplished without knowledge. That is, a dictionary is insufficient; instead, we would need an encyclopedia.

Realizing that machine translation may be too ambitious, researchers tackled less ambitious applications of NLP in the late 1960s and 1970s with some success, though the techniques developed failed to scale up, thus only having limited application impact. For example, people looked at speech recognition applications where the goal is to transcribe a speech. Such a task requires only limited understanding of natural language, thus more realistic; for example, figuring out the exact syntactic structure is probably not very crucial for speech recognition. Two interesting projects that demonstrated clear ability of computer understanding of natural language are worth mentioning. One is the Eliza project where shallow rules are used to enable a computer to play the role of a therapist to engage a natural language dialogue with a human. The other is the block world project which demonstrated feasibility of deep semantic understanding of natural language when the language is limited to a toy domain with only blocks as objects.

In the 1970s–1980s, attention was paid to process real-world natural-language text data, particularly story understanding. Many formalisms for knowledge representation and heuristic inference rules were developed. However, the general conclusion was that even simple stories are quite challenging to understand by a computer, confirming the need for large-scale knowledge representation and inferences under uncertainty.

After the 1980s, researchers started moving away from the traditional symbolic (logic-based) approaches to natural language processing, which mostly had proven to be not robust for real applications, and paying more attention to statistical approaches, which enjoyed more success, initially in speech recognition, but later also in virtually all other NLP tasks. In contrast to symbolic approaches, statistical approaches tend to be more robust because they have less reliance on human-generated rules; instead, they often take advantage of regularities and patterns in

empirical uses of language, and rely solely on labeled training data by humans and application of machine learning techniques.

While linguistic knowledge is always useful, today, the most advanced natural language processing techniques tend to rely on heavy use of statistical machine learning techniques with linguistic knowledge only playing a somewhat secondary role. These statistical NLP techniques are successful for some of the NLP tasks. Part of speech tagging is a relatively easy task, and state-of-the-art POS taggers may have a very high accuracy (above 97% on news data). Parsing is more difficult, though partial parsing can probably be done with reasonably high accuracy (e.g., above 90% for recognizing noun phrases)[1].

However, full structure parsing remains very difficult, mainly because of ambiguities. Semantic analysis is even more difficult, only successful for some aspects of analysis, notably information extraction (recognizing named entities such as names of people and organization, and relations between entities such as who works in which organization), word sense disambiguation (distinguishing different senses of a word in different contexts of usage), and sentiment analysis (recognizing positive opinions about a product in a product review). Inferences and speech act analysis are generally only feasible in very limited domains.

In summary, only "shallow" analysis of natural language processing can be done for arbitrary text and in a robust manner; "deep" analysis tends not to scale up well or be robust enough for analyzing unrestricted text. In many cases, a significant amount of training data (created by human labeling) must be available in order to achieve reasonable accuracy.

3.2 NLP and Text Information Systems

Because of the required robustness and efficiency in TIS applications, in general, robust shallow NLP techniques tend to be more useful than fragile deep analysis techniques, which may hurt application performance due to inevitable analysis errors caused by the general difficulty of NLP. The limited value of deep NLP for some TIS tasks is further due to various ways to bypass the more difficult task of precisely understanding the meaning of natural language text and directly optimize the task performance. Thus, while improved NLP techniques should in general enable improved TIS task performance, lack of NLP capability isn't necessarily a major barrier for some application tasks, notably text retrieval, which is a relatively

1. These performance numbers were based on a specific data set, so they may not generalize well even within the same domain.

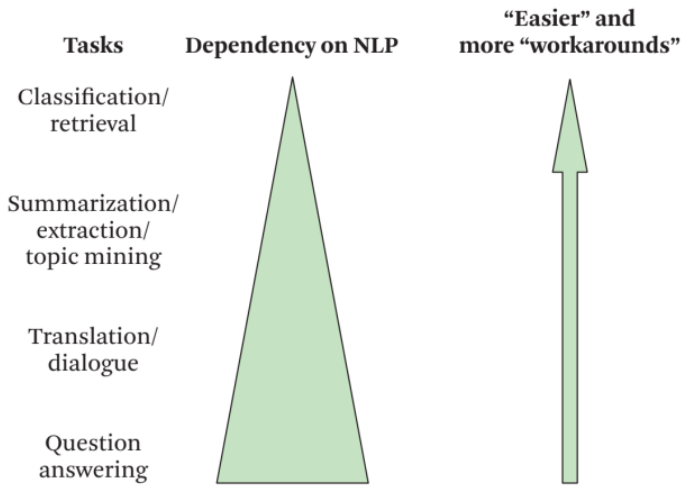

Figure 3.2 **"Easy" vs. "difficult" NLP applications.**

easy task as compared with a more difficult task such as machine translation where deep understanding of natural language is clearly required.

Figure 3.2 shows a number of TIS tasks that require somewhat different levels of NLP. At one end of the spectrum, tasks such as retrieval and classification are relatively easy, and in most of the cases, they don't require deep NLP; indeed, looking at the keywords mentioned in text is often sufficient to determine whether a document is relevant to a query or about a certain topic. At the other end, however, tasks such as machine translation and question answering would require much more precise understanding; for example, a wrong parse of a sentence generally would lead to a wrong translation unless the target language has a similar ambiguity structure, and similarly, a wrong understanding of the question would lead to wrong answers.

When it comes to a specific application task, it is often possible to bypass the difficulty in accurately understanding natural language and go directly to solve the application problem. A well-known example is the Eliza system,[2] which is supposed to play the role of a therapist and make a dialogue with a human user [Weizenbaum 1966]. The following is a sample dialogue.

Person: Men are all alike.

2. http://www.cs.nott.ac.uk/~pszgxk/courses/g5aiai/002history/eliza.htm

ELIZA : In what way?

Person: They're always bugging us about something or other.

ELIZA: Can you think of a specific example?

Person: Well, my boyfriend made me come here.

ELIZA: Your boyfriend made you come here?

Person: He says I'm depressed much of the time.

ELIZA: I am sorry to hear that you are depressed.

Person: It's true. I am unhappy.

ELIZA: Do you think coming here will help you not to be unhappy?

Person: I need some help, that much seems certain.

ELIZA: What would it mean to you if you got some help?

Person: Perhaps I could learn to get along with my mother.

ELIZA: Tell me more about your family

On the surface, the dialogue appears to be quite natural, and indeed, such a dialogue might be useful to engage a depressed patient in a conversation. However, the system does not really understand the language, and solely relies on heuristic rules like the following to keep the dialogue going:

$$\text{I remember } X \rightarrow \text{Do you often think of } X?$$

$$\text{always} \rightarrow \text{Can you think of a specific example?}$$

Such rules enable the system to directly perform the task, i.e., making a conversation, without necessarily trying to understand the real meaning of words and determining the meaning of the entire sentence.

Such a pattern-based way of solving a problem has turned out to be quite powerful. Indeed, modern machine learning approaches to natural language understanding are essentially based on this and in many ways are similar to the Eliza system, but with two important differences. The first is that the rules in a machine learning system would not be exact or strict; instead, they tend to be stochastic, and the probabilities of choosing which rule would be empirically set based on a training data set where the expected behavior of a function to be computed is known. Second, instead of having human to supply rules, the "soft" rules may be learned

automatically from the training data with only minimum help from users who can, e.g., specify the elements in a rule.

Even difficult tasks like machine translation can be done by such statistical approaches. The most useful NLP techniques for building a TIS are statistical approaches which tend to be much more robust than symbolic approaches. Statistical language models are especially useful because they can quantify the uncertainties associated with the use of natural language in a principled way.

3.3 Text Representation

Techniques from NLP allow us to design many different types of informative features for text objects. Let's take a look at the example sentence *A dog is chasing a boy on the playground* in Figure 3.3. We can represent this sentence in many different ways. First, we can always represent such a sentence as a string of characters. This is true for every language. This is perhaps the most general way of representing text since we can always use this approach to represent any text data. Unfortunately, the downside to this representation is that it can't allow us to perform semantic analysis, which is often needed for many applications of text mining. We're not even recognizing words, which are the basic units of meaning for any language. (Of course, there are some situations where characters are useful, but that is not the general case.)

The next version of text representation is performing word segmentation to obtain a sequence of words. In the example sentence, we get features like *dog* and *chasing*. With this level of representation, we suddenly have much more freedom. By identifying words, we can (for example), easily discover the most frequent words in this document or the whole collection. These words can then be used to form topics. Therefore, representing text data as a sequence of words opens up a lot of interesting analysis possibilities.

However, this level of representation is slightly less general than a string of characters. In some languages, such as Chinese, it's actually not that easy to identify all the word boundaries since in such a language text is a sequence of characters with no spaces in between words. To solve this problem, we have to rely on some special techniques to identify words and perform more advanced segmentation that isn't only based on whitespace (which isn't always 100% accurate). So, the sequence of words representation is not as robust as the string of characters representation. In English, it's very easy to obtain this level of representation so we can use this all the time.

If we go further in natural language processing, we can add part-of-speech (POS) tags to the words. This allows us to count, for example, the most frequent nouns; or,

we could determine what kind of nouns are associated with what kind of verbs. This opens up more interesting opportunities for further analysis. Note in Figure 3.3 that we use a plus sign on the additional features because by representing text as a sequence of part of speech tags, we don't necessarily replace the original word sequence. Instead, we add this as an additional way of representing text data.

Representing text as both words and POS tags enriches the representation of text data, enabling a deeper, more principled analysis. If we go further, then we'll be parsing the sentence to obtain a syntactic structure. Again, this further opens up more interesting analysis of, for example, the writing styles or grammatical error correction.

If we go further still into semantic analysis, then we might be able to recognize *dog* as an animal. We also can recognize *boy* as a person, and *playground* as a location and analyze their relations. One deduction could be that the dog was chasing the boy, and the boy is on the playground. This will add more entities and relations, through entity-relation recognition. Now, we can count the most frequent person that appears in this whole collection of news articles. Or, whenever you see a mention of this person you also tend to see mentions of another person or object. These types of repeated pattens can potentially make very good features.

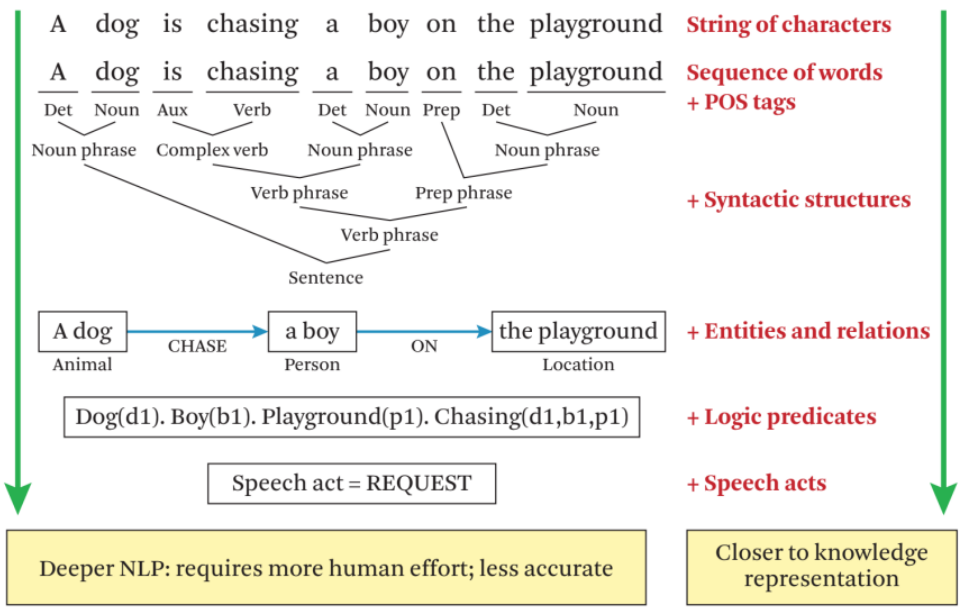

Figure 3.3 **Illustration of different levels of text representation.**

Such a high-level representation is even less robust than the sequence of words or POS tags. It's not always easy to identify all the entities with the right types and we might make mistakes. Relations are even harder to find; again, we might make mistakes. The level of representation is less robust, yet it's very useful. If we move further to a logic representation, then we have predicates and inference rules. With inference rules we can infer interesting derived facts from the text. As one would imagine, we can't do that all the time for all kinds of sentences since it may take significant computation time or a large amount of training data.

Finally, speech acts would add yet another level of representation of the intent of this sentence. In this example, it might be a request. Knowing that would allow us to analyze even more interesting things about the observer or the author of this sentence. What's the intention of saying that? What scenarios or what kinds of actions will occur?

Figure 3.3 shows that if we move downwards, we generally see more sophisticated NLP techniques. Unfortunately, such techniques would require more human effort as well, and they are generally less robust since they attempt to solve a much more difficult problem. If we analyze our text at levels that represent deeper analysis of language, then we have to tolerate potential errors. That also means it's still necessary to combine such deep analysis with shallow analysis based on (for example) sequences of words. On the right side, there is an arrow that points down to indicate that as we go down, our representation of text is closer to the knowledge representation in our mind. That's the purpose of text mining!

Clearly, there is a tradeoff here between doing deeper analysis that might have errors but would give us direct knowledge that can be extracted from text and doing shadow analysis that is more robust but wouldn't give us the necessary deeper representation of knowledge.

Text data are generated by humans and are meant to be consumed by humans. As a result, in text data analysis and text mining, humans play a very important role. They are always in the loop, meaning that we should optimize for a collaboration between humans and computers. In that sense, it's okay that computers may not be able to have a completely accurate representation of text data. Patterns that are extracted from text data can be interpreted by humans, and then humans can guide the computers to do more accurate analysis by annotating more data, guiding machine learning programs to make them work more effectively.

Different text representation tends to enable different analyses, as shown in Figure 3.4. In particular, we can gradually add more and more deeper analysis results to represent text data that would open up more interesting representation opportunities and analysis capabilities. The table summarizes what we have just

Text Rep	Generality	Enabled Analysis	Examples of Application
String		String processing	Compression
Words		Word relation analysis; topic analysis; sentiment analysis	Thesaurus discovery; topic- and opinion-related applications
+ Syntactic structures		Syntactic graph analysis	Stylistic analysis; structure-based feature extraction
+ Entities & relations		Knowledge graph analysis; information network analysis	Discovery of knowledge and opinions about specific entities
+ Logic predicates		Integrative analysis of scattered knowledge; logic inference	Knowledge assistant for biologists

Figure 3.4 **Text representation and enabled analysis.**

seen; the first column shows the type of text representation while the second visualizes the generality of such a representation. By generality, we mean whether we can do this kind of representation accurately for all the text data (very general) or only some of them (not very general). The third column shows the enabled analysis techniques and the final column shows some examples of applications that can be achieved with a particular level of representation.

As a sequence of characters, text can only be processed by string processing algorithms. They are very robust and general. In a compression application, we don't need to know word boundaries (although knowing word boundaries might actually help). Sequences of words (as opposed to characters) offer a very important level of representation; it's quite general and relatively robust, indicating that it supports many analysis techniques such as word relation analysis, topic analysis, and sentiment analysis. As you may expect, many applications can be enabled by these kinds of analysis. For example, thesaurus discovery has to do with discovering related words, and topic- and opinion-related applications can also be based on word-level representation. People might be interested in knowing major topics covered in the collection of text, where a topic is represented as a distribution over words.

Moving down, we'll see we can gradually add additional representations. By adding syntactic structures, we can enable syntactic graph analysis; we can use graph mining algorithms to analyze these syntactic graphs. For example, stylistic

analysis generally requires syntactical structure representation. We can also generate structure-based features that might help us classify the text objects into different categories by looking at their different syntactic structures. If you want to classify articles into different categories corresponding to different authors, then you generally need to look at syntactic structures. When we add entities and relations, then we can enable other techniques such as knowledge graphs or information networks. Using these more advanced feature representations allows applications that deal with entities.

Finally, when we add logical predicates, we can integrate analysis of scattered knowledge. For example, we can add an ontology on top of extracted information from text to make inferences. A good example of an application enabled by this level of representation is a knowledge assistant for biologists. This system is able to manage all the relevant knowledge from literature about a research problem such as understanding gene functions. The computer can make inferences about some of the hypotheses that a biologist might be interested in. For example, it could determine whether a gene has a certain function by reading literature to extract relevant facts. It could use a logic system to track answers to researchers' questions about what genes are related to what functions. In order to support this level of application, we need to go as far as logical representation.

This book covers techniques mainly focused on word-based representation. These techniques are general and robust and widely used in various applications. In fact, in virtually all text mining applications, you need this level of representation. Still, other levels can be combined in order to support more linguistically sophisticated applications as needed.

3.4 Statistical Language Models

A statistical language model (or just language model for short) is a probability distribution over word sequences. It thus gives any sequence of words a potentially different probability. For example, a language model may give the following three-word sequences different probabilities:

$$p(\text{Today is Wednesday}) = 0.001$$

$$p(\text{Today Wednesday is}) = 0.000000001$$

$$p(\text{The equation has a solution}) = 0.000001$$

Clearly, a language model can be context-dependent. In the language model shown above, the sequence *The equation has a solution* has a smaller probability than *Today is Wednesday.* This may be a reasonable language model for describ-

ing general conversations, but it may be inaccurate for describing conversations happening at a mathematics conference, where the sequence *The equation has a solution* may occur more frequently than *Today is Wednesday.*

Given a language model, we can sample word sequences according to the distribution to obtain a text sample. In this sense, we may use such a model to "generate" text. Thus, a language model is also often called a generative model for text.

Why is a language model useful? A general answer is that it provides a principled way to quantify the uncertainties associated with the use of natural language. More specifically, it allows us to answer many interesting questions related to text analysis and information retrieval. The following are some examples of questions that a language model can help answer.

- Given that we see *John* and *feels*, how likely will we see *happy* as opposed to *habit* as the next word? Answering this question can help speech recognition as *happy* and *habit* have very similar acoustic signals, but a language model can easily suggest that *John feels happy* is far more likely than *John feels habit.*
- Given that we observe *baseball* three times and *game* once in a news article, how likely is it about the topic "sports"? This will obviously directly help text categorization and information retrieval tasks.
- Given that a user is interested in sports news, how likely would it be for the user to use *baseball* in a query? This is directly related to information retrieval.

If we enumerate all the possible sequences of words and give a probability to each sequence, the model would be too complex to estimate because the number of parameters is potentially infinite since we have a potentially infinite number of word sequences. That is, we would never have enough data to estimate these parameters. Thus, we have to make assumptions to simplify the model.

The simplest language model is the unigram language model in which we assume that a word sequence results from generating each word independently. Thus, the probability of a sequence of words would be equal to the product of the probability of each word. Formally, let V be the set of words in the vocabulary, and $w_1, \ldots, w_n$ a word sequence, where $w_i \in V$ is a word. We have

$$p(w_1, \ldots, w_n) = \prod_{i=1}^{n} p(w_i). \tag{3.1}$$

Given a unigram language model θ, we have as many parameters as the words in the vocabulary, and they satisfy the constraint $\sum_{w \in V} p(w) = 1$. Such a model essentially specifies a multinomial distribution over all the words.

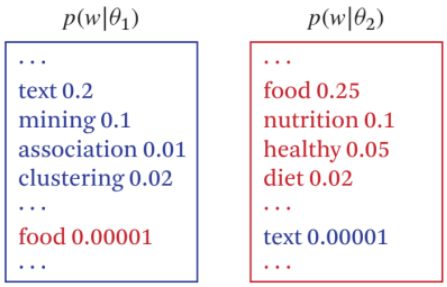

Figure 3.5 **Two examples of unigram language models, representing two different topics.**

Given a language model θ, in general, the probabilities of generating two different documents D_1 and D_2 would be different, i.e., $p(D_1 \mid \theta) \neq p(D_2 \mid \theta)$. What kind of documents would have higher probabilities? Intuitively it would be those documents that contain many occurrences of the high probability words according to $p(w \mid \theta)$. In this sense, the high probability words of θ can indicate the topic captured by θ.

For example, the two unigram language models illustrated in Figure 3.5 suggest a topic about "text mining" and a topic about "health", respectively. Intuitively, if D is a text mining paper, we would expect $p(D \mid \theta_1) > p(D \mid \theta_2)$, while if D' is a blog article discussing diet control, we would expect the opposite: $p(D' \mid \theta_1) < p(D' \mid \theta_2)$. We can also expect $p(D \mid \theta_1) > p(D' \mid \theta_1)$ and $p(D \mid \theta_2) < p(D' \mid \theta_2)$.

Now suppose we have observed a document D (e.g., a short abstract of a text mining paper) which is assumed to be generated using a unigram language model θ, and we would like to infer the underlying model θ (i.e., estimate the probabilities of each word w, $p(w \mid \theta)$) based on the observed D. This is a standard problem in statistics called parameter estimation and can be solved using many different methods.

One popular method is the maximum likelihood (ML) estimator, which seeks a model $\hat{\theta}$ that would give the observed data the highest likelihood (i.e., best explain the data):

$$\hat{\theta} = \arg\max_{\theta} p(D \mid \theta). \tag{3.2}$$

It is easy to show that the ML estimate of a unigram language model gives each word a probability equal to its relative frequency in D. That is,

$$p(w \mid \hat{\theta}) = \frac{c(w, D)}{|D|}, \tag{3.3}$$

where $c(w, D)$ is the count of word w in D and $|D|$ is the length of D, or total number of words in D.

Such an estimate is optimal in the sense that it would maximize the probability of the observed data, but whether it is really optimal for an application is still questionable. For example, if our goal is to estimate the language model in the mind of an author of a research article, and we use the maximum likelihood estimator to estimate the model based only on the abstract of a paper, then it is clearly non-optimal since the estimated model would assign zero probability to any unseen words in the abstract, which would make the whole article have a zero probability unless it only uses words in the abstract. Note that, in general, the maximum likelihood estimate would assign zero probability to any unseen token or event in the observed data; this is so because assigning a non-zero probability to such a token or event would take away probability mass that could have been assigned to an observed word (since all probabilities must sum to 1), thus reducing the likelihood of the observed data. We will discuss various techniques for improving the maximum likelihood estimator later by using techniques called **smoothing**.

Although extremely simple, a unigram language model is already very useful for text analysis. For example, Figure 3.6 shows three different unigram language models estimated on three different text data samples, i.e., a general English text database, a computer science research article database, and a text mining research paper. In general, the words with the highest probabilities in all the three models are those functional words in English because such words are frequently used in any text. After going further down on the list of words, one would see more content-carrying and topical words. Such content words would differ dramatically depending on the data to be used for the estimation, and thus can be used to discriminate the topics in different text samples.

Unigram language models can also be used to perform semantic analysis of word relations. For example, we can use them to find what words are semantically associated with a word like *computer*. The main idea for doing this is to see what other words tend to co-occur with the word *computer*. Specifically, we can first obtain a sample of documents (or sentences) where *computer* is mentioned. We can then estimate a language model based on this sample to obtain $p(w \mid \text{computer})$. This model tells us which words occur frequently in the context of "computer." However, the most frequent words according to this model would likely be functional words in English or words that are simply common in the data, but have no strong association with *computer*. To filter out such common words, we need a model for such words which can then tell us what words should be filtered. It is easy to see that the general English language model (i.e., a background language model) would serve

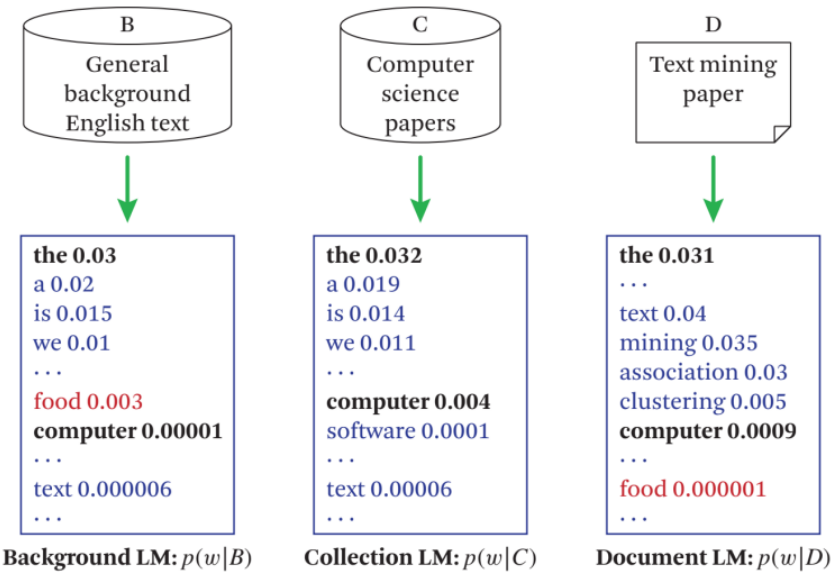

Figure 3.6 **Three different language models representing three different topics.**

the purpose well. So we can use the background language model to normalize the model $p(w \mid \text{computer})$ and obtain a probability ratio for each word. Words with high ratio values can then be assumed to be semantically associated with *computer* since they tend to occur frequently in its context, but not frequently in general. This is illustrated in Figure 3.7.

More applications of language models in text information systems will be further discussed as their specific applications appear in later chapters. For example, we can represent both documents and queries as being generated from some language model. Given this background however, the reader should have sufficient information to understand the future chapters dealing with this powerful statistical tool.

Bibliographic Notes and Further Reading

There are many textbooks on NLP, including, *Speech and Language Processing* [Jurafsky and Martin 2009], *Foundations of Statistical NLP* [Manning and Schütze 1999], and *Natural Language Understanding* [Allen 1995]. An in-depth coverage of statistical language models can be found in the book *Statistical Methods for Speech Recognition* [Jelinek 1997]. Rosenfeld [2000] provides a concise yet comprehensive

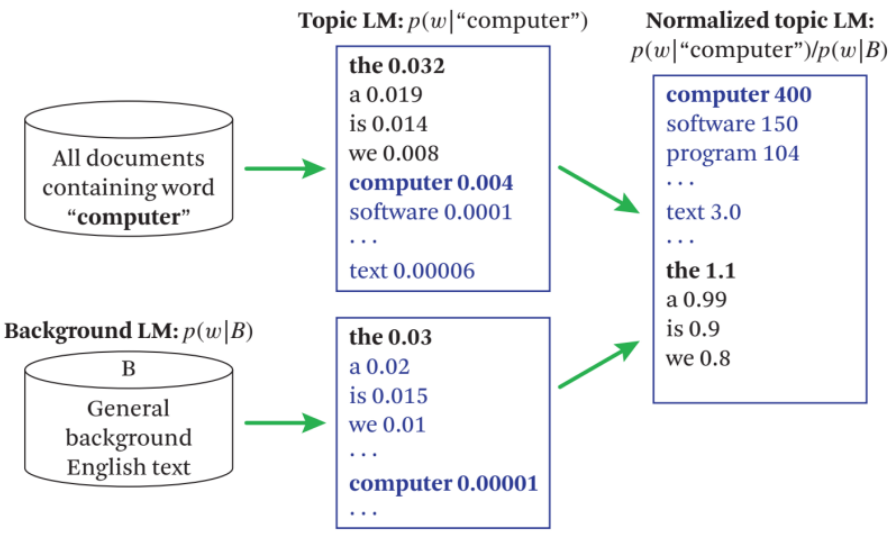

Figure 3.7 **Using topic language models and a background language model to find semantically related words.**

review of statistical language models. Zhai [2008] contains a detailed discussion of the use of statistical language models for information retrieval, some of which will be covered in later chapters of this book. An important topic in NLP that we have not covered much in this chapter is information extraction. A comprehensive introduction to this topic can be found in Sarawagi [2008], and a useful survey can be found in Jiang [2012]. For a discussion of this topic in the context of information retrieval, see the book Moens [2006].

Exercises

3.1. In what way is NLP related to text mining?

3.2. Does poor NLP performance mean poor retrieval performance? Explain.

3.3. Given a collection of documents for a specific topic, how can we use maximum likelihood estimation to create a topic unigram language model?

3.4. How might the size of a document collection affect the quality of a language model?

3.5. Why might maximum likelihood estimation not be the best guess of parameters for a topic language model?

3.6. Suppose we appended a duplicate copy of the topic collection to itself and re-estimated a maximum likelihood language model. Would θ change?

3.7. A unigram language model as defined in this chapter can take a sequence of words as input and output its probability. Explain how this calculation has strong independence assumptions.

3.8. Given a unigram language model θ estimated from this book, and two documents $d_1 =$ `information retrieval` and $d_2 =$ `retrieval information`, then is $p(d_1 \mid \theta) > p(d_2 \mid \theta)$? Explain.

3.9. An n-gram language model records sequences of n words. How does the number of possible parameters change if we decided to use a 2-gram (bigram) language model instead of a unigram language model? How about a 3-gram (trigram) model? Give your answer in terms of V, the unigram vocabulary size.

3.10. Using your favorite programming language, estimate a unigram language model using maximum likelihood. Do this by reading a single text file and delimiting words by whitespace.

3.11. Sort the words by their probabilities from the previous exercise. If you used a different text file, how would your sorted list be different? How would it be the same?

4 META: A Unified Toolkit for Text Data Management and Analysis

This chapter introduces the accompanying software META, a free and open-source toolkit that can be used to analyze text data. Throughout this book, we give hands-on exercises with META to practice concepts and explore different text mining algorithms.

Most of the algorithms and methods discussed in this book can be found in some form in the META toolkit. If META doesn't include a technique discussed in this book, it's likely that a chapter exercise is to implement this feature yourself! Despite being a powerful toolkit, META's simplicity makes feature additions relatively straightforward, usually through extending a short class hierarchy.

Configuration files are an integral part of META's forward-facing infrastructure. They are designed such that exploratory analysis usually requires no programming effort from the user. By default, META is packaged with various executables that can be used to solve a particular task. For example, for a classification experiment the user would run the following command in their terminal[1]:

```
./classify config.toml
```

This is standard procedure for using the default executables; they take only one configuration file parameter. The configuration file format is explained in detail later in this chapter, but essentially it allows the user to select a dataset, a way

1. Running the default classification experiment requires a dataset to operate on. The 20newsgroups dataset is specified in the default META config file and can be downloaded here: https://meta-toolkit.org/data/20newsgroups.tar.gz. Place it in the `meta/data/` directory.

to tokenize the dataset, and a particular classification algorithm to run (for this example).

If more advanced functionality is desired, programming in C++ is required to make calls to META's API (applications programming interface). Both configuration file and API usage are documented on META's website, https://meta-toolkit.org as well as in this chapter. Additionally, a forum for META exists (https://forum.meta-toolkit.org), containing discussion surrounding the toolkit. It includes user support topics, community-written documentation, and developer discussions.

The sections that follow delve into a little more detail about particular aspects of META so the reader will be comfortable working with it in future chapters.

4.1 Design Philosophy

META's goal is to improve upon and complement the current body of open source machine learning and information retrieval software. The existing environment of this open source software tends to be quite fragmented.

There is rarely a single location for a wide variety of algorithms; a good example of this is the LIBLINEAR [Fan et al. 2008] software package for SVMs. While this is the most cited of the open source implementations of linear SVMs, it focuses solely on kernel-less methods. If presented with a nonlinear classification problem, one would be forced to find a different software package that supports kernels (such as the same authors' LIBSVM package [Chang and Lin 2011]). This places an undue burden on the researchers and students—not only are they required to have a detailed understanding of the research problem at hand, but they are now forced to understand this fragmented nature of the open-source software community, find the appropriate tools in this mishmash of implementations, and compile and configure the appropriate tool.

Even when this is all done, there is the problem of data formatting—it is unlikely that the tools have standardized upon a single input format, so a certain amount of data preprocessing is now required. This all detracts from the actual task at hand, which has a marked impact on the speed of discovery and education.

META addresses these issues. In particular, it provides a unifying framework for text indexing and analysis methods, allowing users to quickly run controlled experiments. It modularizes the feature generation, instance representation, data storage formats, and algorithm implementations; this allows for researchers and students to make seamless transitions along any of these dimensions with minimal effort.

META's modularity supports exploration, encourages contributions, and increases visibility to its inner workings. These facts make it a perfect companion toolkit for this book. As mentioned at the beginning of the chapter, readers will follow exercises that add real functionality to the toolkit. After reading this book and learning about text data management and analysis, it is envisioned readers continue to modify META to suit their text information needs, building upon their newfound knowledge.

Finally, since META will always be free and open-source, readers as a community can jointly contribute to its functionality, benefiting all those involved.

4.2 Setting up META

All future sections in this book will assume the reader has META downloaded and installed. Here, we'll show how to set up META.

META has both a website with tutorials and an online repository on GitHub. To actually download the toolkit, users will check it out with the version control software `git` in their command line terminal after installing any necessary prerequisites.

The META website contains instructions for downloading and setting up the software for a particular system configuration. At the time of writing this book, both Linux and Mac OS are supported. Visit https://meta-toolkit.org/setup-guide.html and follow the instructions for the desired platform. We will assume the reader has performed the steps listed in the setup guide and has a working version of META for all exercises and demonstrations in this book.

There are two steps that are not mentioned in the setup guide. The first is to make sure the reader has the version of META that was current when this book was published. To ensure that the commands and examples sync up with the software the reader has downloaded, we will ensure that META is checked out with version 2.2.0. Run the following command inside the `meta/` directory:

```
git reset --hard v2.2.0
```

The second step is to make sure that any necessary model files are also downloaded. These are available on the META releases page: https://github.com/meta-toolkit/meta/releases/tag/v2.2.0. By default, the model files should be placed in the `meta/build/` directory, but you can place them anywhere as long as the paths in the config file are updated.

Once these steps are complete, the reader should be able to complete any exercise or run any demo. If any additional files or information are needed, it will be provided in the accompanying section.

4.3 Architecture

All processed data in META is stored in an index. There are two index types: `forward_index` and `inverted_index`. The former is keyed by document IDs, and the latter is keyed by term IDs.

`forward_index` is used for applications such as topic modeling and most classification tasks.

`inverted_index` is used to create search engines, or do classification with *k*-nearest-neighbor or similar algorithms.

Since each META application takes an index as input, all processed data is interchangeable between all the components. This also gives a great advantage to classification: META supports out-of-core classification by default! If a dataset is small enough (like most other toolkits assume), a cache can be used such as `no_evict_cache` to keep it all in memory without sacrificing any speed. (Index usage is explained in much more detail in the search engine exercises.)

There are four corpus input formats.

`line_corpus`. each dataset consists of one to three files:

`corpusname.dat`. each document appears on one line

`corpusname.dat.labels`. optional file that includes the class or label of the document on each line, again corresponding to the order in `corpusname.dat`. These are the labels that are used for the classification tasks.

`file_corpus`. each document is its own file, and the name of the file becomes the name of the document. There is also a `corpusname-full-corpus.txt` which contains (on each line) a required class label for each document followed by the path to the file on disk. If there are no class labels, a placeholder label should be required, e.g., "[none]".

`gz_corpus`. similar to `line_corpus`, but each of its files and metadata files are compressed using gzip compression:

`corpusname.dat.gz`. compressed version of `corpusname.dat`

`corpusname.dat.labels.gz`. compressed version of `corpusname.dat.labels`

`libsvm_corpus`. If only being used for classification, MeTA can also take LIBSVM-formatted input to create a `forward_index`. There are many machine learning datasets available in this format on the LIBSVM site.[2]

For more information on corpus storage and configuration settings, we suggest the reader consult https://meta-toolkit.org/overview-tutorial.html.

4.4 Tokenization with MeTA

The first step in creating an index over any sort of text data is the "tokenization" process. At a high level, this simply means converting individual text documents into sparse vectors of counts of terms—these sparse vectors are then typically consumed by an indexer to output an `inverted_index` over your corpus.

MeTA structures this text analysis process into several layers in order to give the user as much power and control over the way the text is analyzed as possible.

An analyzer, in most cases, will take a "filter chain" that is used to generate the final tokens for its tokenization process: the filter chains are always defined as a specific tokenizer class followed by a sequence of zero or more filter classes, each of which reads from the previous class's output. For example, here is a simple filter chain that lowercases all tokens and only keeps tokens with a certain length range:

`icu_tokenizer` → `lowercase_filter` → `length_filter`

Tokenizers always come first. They define how to split a document's string content into tokens. Some examples are as follows.

`icu_tokenizer`. converts documents into streams of tokens by following the Unicode standards for sentence and word segmentation.

`character_tokenizer`. converts documents into streams of single characters.

Filters come next, and can be chained together. They define ways that text can be modified or transformed. Here are some examples of filters.

`length_filter`. this filter accepts tokens that are within a certain length and rejects those that are not.

`icu_filter`. applies an ICU (International Components for Unicode)[3] transliteration to each token in the sequence. For example, an accented character like $\ddot{i}$ is instead written as i.

2. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets

3. http://site.icu-project.org/; note that different versions of ICU will tokenize text in slightly different ways!

`list_filter`. this filter either accepts or rejects tokens based on a list. For example, one could use a stop word list and reject stop words.

`porter2_stemmer`. this filter transforms each token according to the Porter2 English Stemmer rules.[4]

Analyzers operate on the output from the filter chain and produce token counts from documents. Here are some examples of analyzers.

`ngram_word_analyzer`. Collects and counts sequences of n words (tokens) that have been filtered by the filter chain.

`ngram_pos_analyzer`. Same as `ngram_word_analyzer`, but operates on part-of-speech tags from META's CRF implementation.

`tree_analyzer`. Collects and counts occurrences of parse tree features.

`libsvm_analyzer`. Converts a LIBSVM `line_corpus` into META format.

META defines a sane default filter chain that users are encouraged to use for general text analysis in the absence of any specific requirements. To use it, one should specify the following in the configuration file:

```
[[analyzers]]
method = "ngram-word"
ngram = 1
filter = "default-chain"
```

This configures the text analysis process to consider unigrams of words generated by running each document through the default filter chain. This filter chain should work well for most languages, as all of its operations (including but not limited to tokenization and sentence boundary detection) are defined in terms of the Unicode standard wherever possible.

To consider both unigrams and bigrams, the configuration file should look like the following:

```
[[analyzers]]
method = "ngram-word"
ngram = 1
filter = "default-chain"

[[analyzers]]
```

4. http://snowball.tartarus.org/algorithms/english/stemmer.html

```
method = "ngram-word"
ngram = 2
filter = "default-chain"
```

Each `[[analyzers]]` block defines a single analyzer and its corresponding filter chain: as many can be used as desired—the tokens generated by each analyzer specified will be counted and placed in a single sparse vector of counts. This is useful for combining multiple different kinds of features together into your document representation. For example, the following configuration would combine unigram words, bigram part-of-speech tags, tree skeleton features, and subtree features.

```
[[analyzers]]
method = "ngram-word"
ngram = 1
filter = "default-chain"

[[analyzers]]
method = "ngram-pos"
ngram = 2
filter = [{type = "icu-tokenizer"}, {type = "ptb-normalizer"}]
crf-prefix = "path/to/crf/model"

[[analyzers]]
method = "tree"
filter = [{type = "icu-tokenizer"}, {type = "ptb-normalizer"}]
features = ["skel", "subtree"]
tagger = "path/to/greedy-tagger/model"
parser = "path/to/sr-parser/model"
```

If an application requires specific text analysis operations, one can specify directly what the filter chain should look like by modifying the configuration file. Instead of filter being a string parameter as above, we will change filter to look very much like the `[[analyzers]]` blocks: each analyzer will have a series of `[[analyzers.filter]]` blocks, each of which defines a step in the filter chain. All filter chains must start with a tokenizer. Here is an example filter chain for unigram words like the one at the beginning of this section:

```
[[analyzers]]
method = "ngram-word"
ngram = 1
    [[analyzers.filter]]
    type = "icu-tokenizer"
```

```
[[analyzers.filter]]
type = "lowercase"

[[analyzers.filter]]
type = "length"
min = 2
max = 35
```

META provides many different classes to support building filter chains. Please look at the API documentation[5] for more information. In particular, the `analyzers::tokenizers` namespace and the `analyzers::filters` namespace should give a good idea of the capabilities. The static public attribute `id` for a given class is the string needed for the "type" in the configuration file.

4.5 Related Toolkits

Existing toolkits supporting text management and analysis tend to fall into two categories. The first is search engine toolkits, which are especially suitable for building a search engine application, but tend to have limited support for text analysis/mining functions. Examples include the following.

Lucene. https://lucene.apache.org/

Terrier. http://terrier.org/

Indri/Lemur. http://www.lemurproject.org/

The second is text mining or general data mining and machine learning toolkits, which tend to selectively support some text analysis functions, but generally do not support search capability. Examples include the following.

Weka. http://www.cs.waikato.ac.nz/ml/weka/

LIBSVM. https://www.csie.ntu.edu.tw/c̆jlin/libsvm/

Stanford NLP. http://nlp.stanford.edu/software/corenlp.shtml

Illinois NLP Curator. http://cogcomp.cs.illinois.edu/page/software_view/Curator

Scikit Learn. http://scikit-learn.org/stable/

NLTK. http://www.nltk.org/

5. Visit https://meta-toolkit.org/doxygen/namespaces.html

However, there is a lack of seamless integration of search engine capabilities with various text analysis functions, which is necessary for building a unified system for supporting text management and analysis. A main design philosophy of MeTA, which also differentiates MeTA from the existing toolkits, is its emphasis on the tight integration of search capabilities (indeed, text access capabilities in general) with text analysis functions, enabling it to provide full support for building a powerful text analysis application. To facilitate education and research, MeTA is designed with an emphasis on modularity and extensibility achieved through object-oriented design. MeTA can be used together with existing toolkits in multiple ways. For example, for very large-scale text applications, an existing search engine toolkit can be used to support search, while MeTA can be used to further support analysis of the found search results or any subset of text data that are obtained from the original large data set. NLP toolkits can be used to preprocess text data and generate annotated text data for modules in MeTA to use as input. MeTA can also be used to generate a text representation that would be fed into a different data mining or machine learning toolkit.

Exercises

In its simplest form, text data could be a single document in `.txt` format. This exercise will get you familiar with various techniques that are used to analyze text. We'll use the novel *A Tale of Two Cities* by Charles Dickens as example text. The book is called `two-cities.txt`, and is located at http://sifaka.cs.uiuc.edu/ir/textdatabook/two-cities.txt. You can also use any of your own plaintext files that have multiple English sentences.

Like all future exercises, we will assume that the reader followed the MeTA setup guide and successfully compiled the executables. In this exercise, we'll only be using the `profile` program. Running `./profile` from inside the `build/` directory will print out the following usage information:

```
Usage: ./profile config.toml file.txt [OPTION]
where [OPTION] is one or more of:
        --stem  perform stemming on each word
        --stop  remove stop words
        --pos   annotate words with POS tags
        --pos-replace   replace words with their POS tags
        --parse create grammatical parse trees from file content
        --freq-unigram  sort and count unigram words
        --freq-bigram   sort and count bigram words
        --freq-trigram  sort and count trigram words
        --all   run all options
```

If running `./profile` prints out this information, then everything has been set up correctly. We'll look into what each of these options mean in the following exercises.

4.1. Stop Word Removal. Consider the following words: *I, the, of, my, it, to, from*. If it was known that a document contained these words, would there be any idea what the document was about? Probably not. These types of words are called stop words. Specifically, they are very high frequency words that do not contain content information. They are used because they're grammatically required, such as when connecting sentences.

Since these words do not contain any topical information, they are often removed as a preprocessing step in text analysis. Not only are these (usually) useless words ignored, but having less data can mean that algorithms run faster!

```
./profile config.toml two-cities.txt --stop
```

Now, use the `profile` program to remove stop words from the document `two-cities.txt`. Can you still get an idea of what the book is about without these words present?

4.2. Stemming. Stemming is the process of reducing a word to a base form. This is especially useful for search engines. If a user wants to find books about *running*, documents containing the word *run* or *runs* would not match. If we apply a stemming algorithm to a word, it is more likely that other forms of the word will match it in an information retrieval task.

The most popular stemming algorithm is the Porter2 English Stemmer, developed by Martin Porter. It is a slightly improved version from the original Porter Stemmer from 1980. Some examples are:

$$\{run, runs, running\} \rightarrow run$$

$$\{argue, argued, argues, arguing\} \rightarrow argu$$

$$\{lies, lying, lie\} \rightarrow lie$$

META uses the Porter2 stemmer by default. You can read more about the Porter2 stemmer here: http://snowball.tartarus.org/algorithms/english/stemmer.html. An online demo of the stemmer is also available if you'd like to play around with it: http://web.engr.illinois.edu/~massung1/p2s_demo.html.

Now that you have an idea of what stemming is, run the stemmer on *A Tale of Two Cities*.

```
./profile config.toml two-cities.txt --stem
```

Like stop word removal, stemming tries to keep the basic meaning behind the original text. Can you still make sense of it after it's stemmed?

4.3. Part-of-Speech Tagging. When learning English, students often encounter different grammatical labels for words, such as *noun, adjective, verb, etc*. In linguistics and computer science, there is a much larger dichotomy of these labels called part of speech (POS) tags. Each word can be assigned a tag based on surrounding words. Consider the following sentence: *All hotel rooms are pretty much the same, although the room number might change*. Here's a part-of-speech tagged version:

$$\begin{aligned}&\text{All}_{DT}\ \text{hotel}_{NN}\ \text{rooms}_{NNS}\ \text{are}_{VBP}\ \text{pretty}_{RB}\ \text{much}_{RB}\ \text{the}_{DT}\ \text{same}_{JJ}\ ,\\&\text{although}_{IN}\ \text{the}_{DT}\ \text{room}_{NN}\ \text{number}_{NN}\ \text{might}_{MD}\ \text{change}_{VB}\ .\end{aligned}$$

Above, VBP and VB are different types of verbs, NN and NNS are singular and plural nouns, and DT means determiner. This is just a subset of about 80 commonly used tags. Not every word has a unique part of speech tag. For instance, *flies* and *like* can have multiple tags depending on the context:

$$\begin{aligned}&\text{Time}_{NN}\ \text{flies}_{VBZ}\ \text{like}_{IN}\ \text{an}_{DT}\ \text{arrow}_{NN}\ .\\&\text{Fruit}_{NN}\ \text{flies}_{NNS}\ \text{like}_{VBP}\ \text{a}_{DT}\ \text{banana}_{NN}\ .\end{aligned}$$

Such situations can make POS-tagging challenging. Nevertheless, human agreement on POS tag labeling is about 97%, which is the ceiling for automatic taggers.

POS tags can be used in text analysis as an alternate (or additional) representation to words. Using these tags captures a slightly more grammatical sense of a document or corpus. The `profile` program has two options for POS tagging. The first annotates each word like the examples above, and the second replaces each word with its POS tag.

```
./profile config.toml two-cities.txt --pos
./profile config.toml two-cities.txt --pos-replace
```

Note that POS tagging the book may take up to one minute to complete. Does it look like META's POS tagger is accurate? Can you find any mistakes? When replacing the words with their tags, is it possible to determine what the original sentence was? Experiment with the book or any other text file.

4.4. Parsing. Grammatical parse trees represent deeper syntactic knowledge from text sentences. They represent sentence phrase hierarchy as a tree structure. Consider the example in Figure 4.1.

The parse tree is rooted with S, denoting Sentence; the sentence is composed of a noun phrase (NP) followed by a verb phrase (VP) and period. The leaves of the

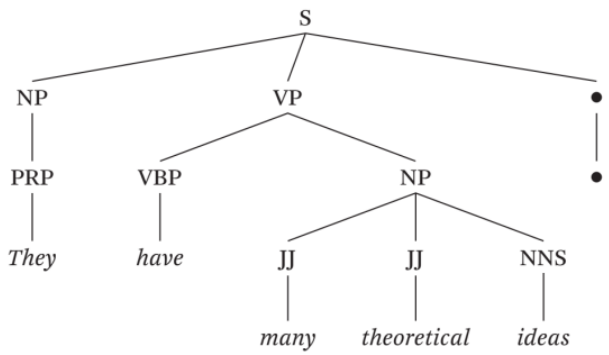

Figure 4.1 **An example of a parse tree.**

tree are the words in the sentence, and the preterminals (the direct parents of the leaves) are part-of-speech tags.

Some common features from a parse tree are production rules such as $S \rightarrow NP\ VP$, tree depth, and structural tree features. Syntactic categories (node labels) alone can also be used.

The following command runs the parser on each sentence in the input file:

```
./profile config.toml two-cities.txt --parse
```

Like POS-tagging, the parsing may also take a minute or two to complete.

4.5. Frequency Analysis. Perhaps the most common text-processing technique is frequency counting. This simply counts how many times each unique word appears in a document (or corpus). Viewing a descending list of words sorted by frequency can give you an idea of what the document is about. Intuitively, similar documents should have some of the same high-frequency words . . . not including stop words.

Instead of single words, we can also look at strings of n words, called n-grams. Consider this sentence: *I took a vacation to go to a beach.*

- 1-grams (unigrams):

$$\{I:1,\ took:1,\ a:2,\ vacation:1,\ to:2,\ go:1,\ beach:1\}$$

- 2-grams (bigrams):

$$\{I\ took:1,\ took\ a:1,\ a\ vacation:1,\ vacation\ to:1,$$
$$to\ go:1,\ go\ to:1,\ to\ a:1,\ a\ beach:1\}$$

- 3-grams (trigrams):

$$\{\textit{I took a}: 1,\ \ \textit{took a vacation}: 1,\ \ \textit{a vacation to}: 1,\ \ \ldots\}$$

As we will see in this text, the unigram words document representation is of utmost importance for text representation. This vector of counts representation does have a downside though: we lost the order of the words. This representation is also known as "bag-of-words," since we only know the counts of each word, and no longer know the context or position. This unigram counting scheme can be used with POS tags or any other type of token derived from a document.

Use the following three commands to do an n-gram frequency analysis on a document, for $n \in [1, 3]$.

```
./profile config.toml two-cities.txt --freq-unigram
./profile config.toml two-cities.txt --freq-bigram
./profile config.toml two-cities.txt --freq-trigram
```

This will give the output file `two-cities.freq.1.txt` for the option `--freq-unigram` and so on.

What makes the output reasonably clear? Think back to stop words and stemming. Removing stop words gets rid of the noisy high-frequency words that don't give any information about the content of the document. Stemming will aggregate inflected words into a single count. This means the partial vector $\{\textit{run}: 4, \textit{running}: 2, \textit{runs}: 3\}$ would instead be represented as $\{\textit{run}: 9\}$. Not only does this make it easier for humans to interpret the frequency analysis, but it can improve text mining algorithms, too!

4.6. Zipf's Law. In English, the top four most frequent words are about 10–15% of all word occurrences. The top 50 words are 35–40% of word occurrences. In fact, there is a similar trend in any human language. Think back to the stop words. These are the most frequent words, and make up a majority of text. At the same time, many words may only appear once in a given document.

We can plot the rank of a word on the x axis, and the frequency count on the y axis. Such a graph can give us an idea of the word distribution in a given document or collection. In Figure 4.2, we counted unigram words from another Dickens book, *Oliver Twist*. The plot on the left is a normal $x \sim y$ plot and the one on the right is a $\log x \sim \log y$ plot.

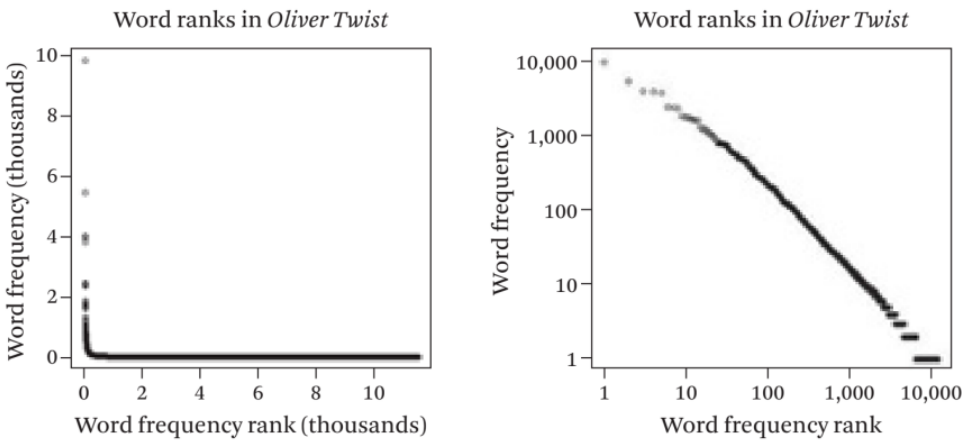

Figure 4.2 **Illustration of Zipf's law.**

Zipf's law describes the shape of these plots. What do you think Zipf's law states? The shape of these plots allows us to apply certain techniques to take advantage of the word distribution in natural language.

PART II

TEXT DATA ACCESS

5 Overview of Text Data Access

Text data access is the foundation for text analysis. Text access technology plays two important roles in text management and analysis applications. First, it enables retrieval of the most relevant text data to a particular analysis problem, thus avoiding unnecessary overhead from processing a large amount of non-relevant data. Second, it enables interpretation of any analysis results or discovered knowledge in appropriate context and provides data provenance (origin).

The general goal of text data access is to connect users with the right information at the right time. This connection can be done in two ways: **pull**, where the users take the initiative to fetch relevant information out from the system, and **push**, where the system takes the initiative to offer relevant information to users. In this chapter, we will give a high-level overview of these two modes of text data access. Then, we will formalize and motivate the problem of text retrieval. In the following chapters, we will cover specific techniques for supporting text access in both push and pull modes.

5.1 Access Mode: Pull vs. Push

Because text data are created for consumption by humans, humans play an important role in text data analysis and management applications. Specifically, humans can help select the most relevant data to a particular application problem, which is beneficial since it enables us to avoid processing the huge amount of raw text data (which would be inefficient) and focus on analyzing the most relevant part. Selecting relevant text data from a large collection is the basic task of text access. This selection is generally based on a specification of the information need of an analyst (a user), and can be done in two modes: pull and push. Figure 5.1 describes how these modes fit together along with querying and browsing.

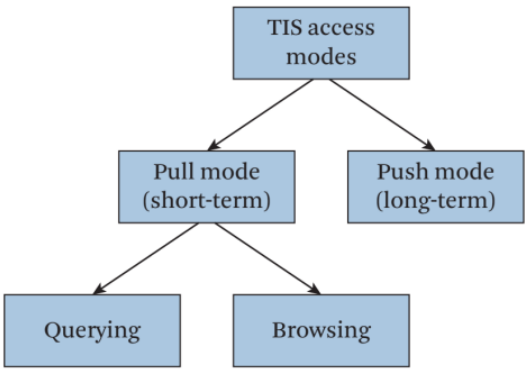

Figure 5.1 **The dichotomy of text information access modes.**

In pull mode, the user initiates the access process to find the relevant text data, typically by using a search engine. This mode of text access is essential when a user has an ad hoc information need, i.e., a temporary information need that might disappear once the need is satisfied. In such a case, the user can use a query to find relevant information with a search engine. For example, a user may have a need to buy a product and thus be interested in retrieving all the relevant opinions about candidate products; after the user has purchased the product, the user would generally no longer need such information. Another example is that during the process of analyzing social media data to understand opinions about an emerging event, the analyst may also decide to explore information about a particular entity related to the event (e.g., a person), which can also trigger a search activity.

While querying is the most common way of accessing text data in the pull mode, browsing is another complementary way of accessing text data in the pull mode, and can be very useful when a user does not know how to formulate an effective query, or finds it inconvenient to enter a keyword query (e.g., through a smartphone), or simply wants to explore a topic with no fixed goal. Indeed, when searching the Web, users tend to mix querying and browsing (e.g., while traversing through hyperlinks).

In general, we may regard querying and browsing as two complementary ways of finding relevant information in the information space. Their relation can be understood by making an analogy between information seeking and sightseeing in a physical world. When a tourist knows the exact address of an attraction, the tourist can simply take a taxi directly to the attraction; this is similar to when a user knows exactly what he or she is looking for and can formulate a query with the

engine interface to enable a user to browse the information space flexibly. With this interface, a user can do any of the following at any moment.

Querying (long-range jump). When a user submits a new query through the search box the search results from a search engine will be shown in the right pane. At the same time, the relevant part of a topic map is also shown on the left pane to facilitate browsing should the user want to.

Navigating on the map (short-range walk). The left pane in our interface is to let a user navigate on the map. When a user clicks on a map node, this pane will be refreshed and a local view with the clicked node as the current focus will be displayed. In the local view, we show the parents, the children, and the horizontal neighbors of the current node in focus (labelled as "center" in the interface). A user can thus zoom into a child node, zoom out to a parent node, or navigate into a horizontal neighbor node. The number attached to a node is a score for the node that we use for ranking the nodes. Such a map enables the user to "walk" in the information space to browse into relevant documents without needing to reformulate queries.

Viewing a topic region. The user may double-click on a topic node on the map to view the documents covered in the topic region. The search result pane would be updated with new results corresponding to the documents in the selected topic region. From a user's perspective, the result pane always shows the documents in the current region that the user is focused on (either search results of the query or the documents corresponding to a current node on the map when browsing).

Viewing a document. Within the result pane, a user can select any document to view as in a standard search interface.

In Figure 5.3, we further show an example trace of browsing in which the user started with a query *dining table*, zoomed into *asian dining table*, zoomed out back to *dining table*, browsed horizontally first to *dining chair* and then to *dining furniture*, and finally zoomed out to the general topic *furniture* where the user would have many options to explore different kinds of furniture. If this user feels that a "long-jump" is needed, he or she can use a new query to achieve it. Since the map can be hidden and only brought to display when the user needs it, such an interface is a very natural extension of the current search interface from a user's perspective. Thus, we can see how one text access system can combine multiple modes of information access to suit a user's current needs.


image
not
available

and would like to find the relevant information immediately. The system to support TR is a text retrieval system, or a search engine.

Although TR is sometimes used interchangeably with the more general term "information retrieval" (IR), the latter also includes retrieval of other types of information such as images or videos. It is worth noting, though, that retrieval techniques for other non-textual data are less mature and, as a result, retrieval of other types of information tends to rely on using text retrieval techniques to match a keyword query with companion text data with a non-textual data element. For example, the current image search engines on the Web are essentially a TR system where each image is represented by a text document consisting of any associated text data with the image (e.g., title, caption, or simply textual context of the image such as the news article content where an image is included).

In industry, the problem of TR is generally referred to as the search problem, and the techniques for text retrieval are often called search technology or search engine technology.

The task of TR can be easy or hard, depending on specific queries and specific collections. For example, during a web search, finding homepages is generally easy, but finding out people's opinions about some topic (e.g., U.S. foreign policy) would be much harder. There are several reasons why TR is difficult:

- a query is usually quite short and incomplete (no formal language like SQL);
- the information need may be difficult to describe precisely, especially when the user isn't familiar with the topic, and
- precise understanding of the document content is difficult. In general, since what counts as the correct answer is subjective, even when human experts judge the relevance of documents, they may disagree with each other.

Due to the lack of clear semantic structures and difficulty in natural language understanding, it is often challenging to accurately retrieve relevant information to a user's query. Indeed, even though the current web search engines may appear to be sufficient sometimes, it may still be difficult for a user to quickly locate and harvest all the relevant information for a task. In general, the current search engines work very well for navigational queries and simple, popular informational queries, but in the case where a user has a complex information need such as analyzing opinions about products to buy, or researching medical information about some symptoms, they often work poorly. Moreover, the current search engines generally provide little or no support to help users digest and exploit the retrieved information. As a result, even if a search engine can retrieve the most relevant information,

Index