CHAPTER 1
NLP: A Primer

A language is not just words. It's a culture, a tradition,
a unification of a community,
a whole history that creates what a community is.
It's all embodied in a language.
—Noam Chomsky

Imagine a hypothetical person, John Doe. He's the CTO of a fast-growing technology startup. On a busy day, John wakes up and has this conversation with his digital assistant:

John: "How is the weather today?"

Digital assistant: "It is 37 degrees centigrade outside with no rain today."

John: "What does my schedule look like?"

Digital assistant: "You have a strategy meeting at 4 p.m. and an all-hands at 5:30 p.m. Based on today's traffic situation, it is recommended you leave for the office by 8:15 a.m."

While he's getting dressed, John probes the assistant on his fashion choices:

John: "What should I wear today?"

Digital assistant: "White seems like a good choice."

You might have used smart assistants such as Amazon Alexa, Google Home, or Apple Siri to do similar things. We talk to these assistants not in a programming language, but in our natural language—the language we all communicate in. This natural language has been the primary medium of communication between humans since time immemorial. But computers can only process data in binary, i.e., 0s and 1s. While we can represent language data in binary, how do we make machines understand the

language? This is where natural language processing (NLP) comes in. It is an area of computer science that deals with methods to analyze, model, and understand human language. Every intelligent application involving human language has some NLP behind it. In this book, we'll explain what NLP is as well as how to use NLP to build and scale intelligent applications. Due to the open-ended nature of NLP problems, there are dozens of alternative approaches one can take to solve a given problem. This book will help you navigate this maze of options and suggests how to choose the best option based on your problem.

This chapter aims to give a quick primer of what NLP is before we start delving deeper into how to implement NLP-based solutions for different application scenarios. We'll start with an overview of numerous applications of NLP in real-world scenarios, then cover the various tasks that form the basis of building different NLP applications. This will be followed by an understanding of language from an NLP perspective and of why NLP is difficult. After that, we'll give an overview of heuristics, machine learning, and deep learning, then introduce a few commonly used algorithms in NLP. This will be followed by a walkthrough of an NLP application. Finally, we'll conclude the chapter with an overview of the rest of the topics in the book. Figure 1-1 shows a preview of the organization of the chapters in terms of various NLP tasks and applications.

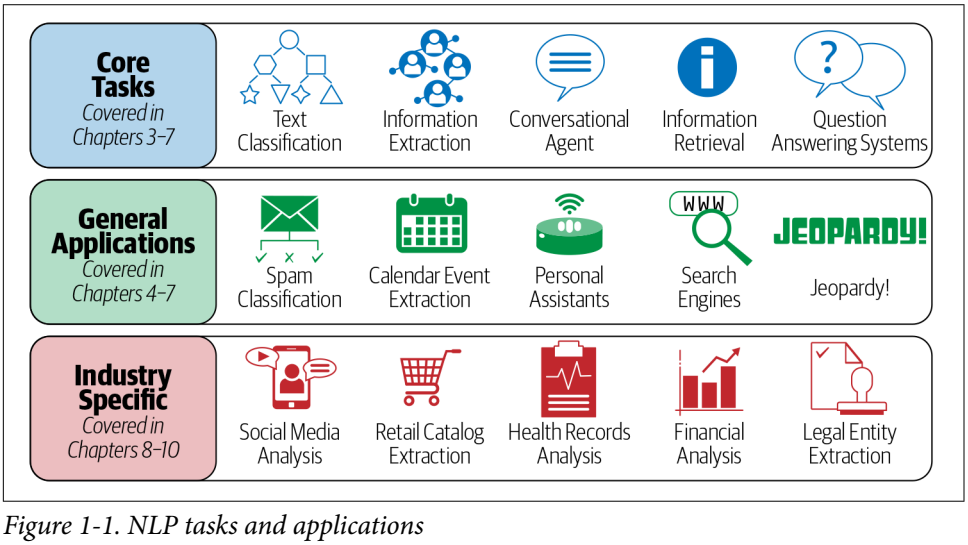

Figure 1-1. NLP tasks and applications

Let's start by taking a look at some popular applications you use in everyday life that have some form of NLP as a major component.

NLP in the Real World

NLP is an important component in a wide range of software applications that we use in our daily lives. In this section, we'll introduce some key applications and also take a look at some common tasks that you'll see across different NLP applications. This section reinforces the applications we showed you in Figure 1-1, which you'll see in more detail throughout the book.

Core applications:

- Email platforms, such as Gmail, Outlook, etc., use NLP extensively to provide a range of product features, such as spam classification, priority inbox, calendar event extraction, auto-complete, etc. We'll discuss some of these in detail in Chapters 4 and 5.
- Voice-based assistants, such as Apple Siri, Google Assistant, Microsoft Cortana, and Amazon Alexa rely on a range of NLP techniques to interact with the user, understand user commands, and respond accordingly. We'll cover key aspects of such systems in Chapter 6, where we discuss chatbots.
- Modern search engines, such as Google and Bing, which are the cornerstone of today's internet, use NLP heavily for various subtasks, such as query understanding, query expansion, question answering, information retrieval, and ranking and grouping of the results, to name a few. We'll discuss some of these subtasks in Chapter 7.
- Machine translation services, such as Google Translate, Bing Microsoft Translator, and Amazon Translate are increasingly used in today's world to solve a wide range of scenarios and business use cases. These services are direct applications of NLP. We'll touch on machine translation in Chapter 7.

Other applications:

- Organizations across verticals analyze their social media feeds to build a better and deeper understanding of the voice of their customers. We'll cover this in Chapter 8.
- NLP is widely used to solve diverse sets of use cases on e-commerce platforms like Amazon. These vary from extracting relevant information from product descriptions to understanding user reviews. Chapter 9 covers these in detail.
- Advances in NLP are being applied to solve use cases in domains such as healthcare, finance, and law. Chapter 10 addresses these.
- Companies such as Arria [1] are working to use NLP techniques to automatically generate reports for various domains, from weather forecasting to financial services.

- NLP forms the backbone of spelling- and grammar-correction tools, such as Grammarly and spell check in Microsoft Word and Google Docs.
- *Jeopardy!* is a popular quiz show on TV. In the show, contestants are presented with clues in the form of answers, and the contestants must phrase their responses in the form of questions. IBM built the Watson AI to compete with the show's top players. Watson won the first prize with a million dollars, more than the world champions. Watson AI was built using NLP techniques and is one of the examples of NLP bots winning a world competition.
- NLP is used in a range of learning and assessment tools and technologies, such as automated scoring in exams like the Graduate Record Examination (GRE), plagiarism detection (e.g., Turnitin), intelligent tutoring systems, and language learning apps like Duolingo.
- NLP is used to build large knowledge bases, such as the Google Knowledge Graph, which are useful in a range of applications like search and question answering.

This list is by no means exhaustive. NLP is increasingly being used across several other applications, and newer applications of NLP are coming up as we speak. Our main focus is to introduce you to the ideas behind building these applications. We do so by discussing different kinds of NLP problems and how to solve them. To get a perspective on what you are about to learn in this book, and to appreciate the nuances that go into building these NLP applications, let's take a look at some key NLP tasks that form the bedrock of many NLP applications and industry use cases.

NLP Tasks

There is a collection of fundamental tasks that appear frequently across various NLP projects. Owing to their repetitive and fundamental nature, these tasks have been studied extensively. Having a good grip on them will make you ready to build various NLP applications across verticals. (We also saw some of these tasks earlier in Figure 1-1.) Let's briefly introduce them:

Language modeling
: This is the task of predicting what the next word in a sentence will be based on the history of previous words. The goal of this task is to learn the probability of a sequence of words appearing in a given language. Language modeling is useful for building solutions for a wide variety of problems, such as speech recognition, optical character recognition, handwriting recognition, machine translation, and spelling correction.

Text classification
: This is the task of bucketing the text into a known set of categories based on its content. Text classification is by far the most popular task in NLP and is used in a variety of tools, from email spam identification to sentiment analysis.

Information extraction
: As the name indicates, this is the task of extracting relevant information from text, such as calendar events from emails or the names of people mentioned in a social media post.

Information retrieval
: This is the task of finding documents relevant to a user query from a large collection. Applications like Google Search are well-known use cases of information retrieval.

Conversational agent
: This is the task of building dialogue systems that can converse in human languages. Alexa, Siri, etc., are some common applications of this task.

Text summarization
: This task aims to create short summaries of longer documents while retaining the core content and preserving the overall meaning of the text.

Question answering
: This is the task of building a system that can automatically answer questions posed in natural language.

Machine translation
: This is the task of converting a piece of text from one language to another. Tools like Google Translate are common applications of this task.

Topic modeling
: This is the task of uncovering the topical structure of a large collection of documents. Topic modeling is a common text-mining tool and is used in a wide range of domains, from literature to bioinformatics.

Figure 1-2 shows a depiction of these tasks based on their relative difficulty in terms of developing comprehensive solutions.

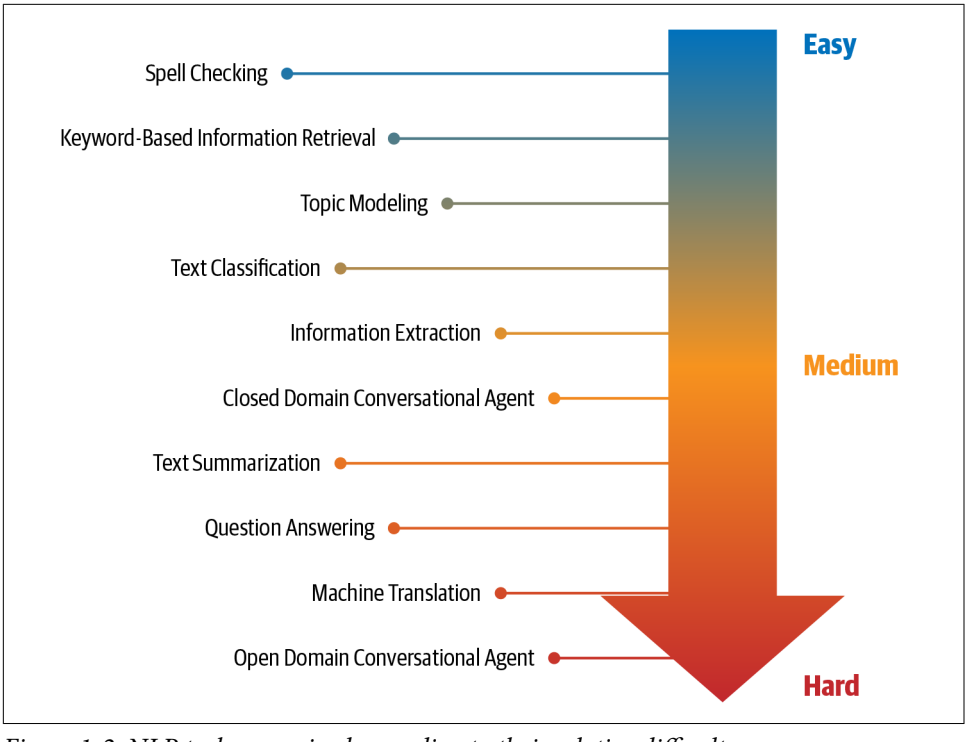

Figure 1-2. NLP tasks organized according to their relative difficulty

In the rest of the chapters in this book, we'll see these tasks' challenges and learn how to develop solutions that work for certain use cases (even the hard tasks shown in the figure). To get there, it is useful to have an understanding of the nature of human language and the challenges in automating language processing. The next two sections provide a basic overview.

What Is Language?

Language is a structured system of communication that involves complex combinations of its constituent components, such as characters, words, sentences, etc. Linguistics is the systematic study of language. In order to study NLP, it is important to understand some concepts from linguistics about how language is structured. In this section, we'll introduce them and cover how they relate to some of the NLP tasks we listed earlier.

We can think of human language as composed of four major building blocks: phonemes, morphemes and lexemes, syntax, and context. NLP applications need knowledge of different levels of these building blocks, starting from the basic sounds of language (phonemes) to texts with some meaningful expressions (context).

Figure 1-3 shows these building blocks of language, what they encompass, and a few NLP applications we introduced earlier that require this knowledge. Some of the terms listed here that were not introduced earlier in this chapter (e.g., parsing, word embeddings, etc.) will be introduced later in these first three chapters.

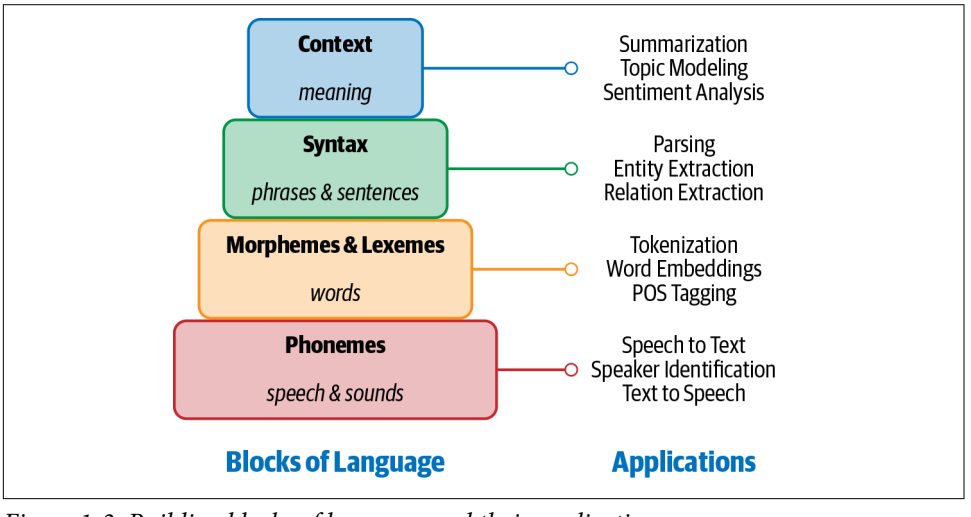

Figure 1-3. Building blocks of language and their applications

Building Blocks of Language

Let's first introduce what these blocks of language are to give context for the challenges involved in NLP.

Phonemes

Phonemes are the smallest units of sound in a language. They may not have any meaning by themselves but can induce meanings when uttered in combination with other phonemes. For example, standard English has 44 phonemes, which are either single letters or a combination of letters [2]. Figure 1-4 shows these phonemes along with sample words. Phonemes are particularly important in applications involving speech understanding, such as speech recognition, speech-to-text transcription, and text-to-speech conversion.

Consonant phonemes, with sample words		Vowel phonemes, with sample words	
1. /b/ - bat	13. /s/ - sun	1. /a/ - ant	13. /oi/ - coin
2. /k/ - cat	14. /t/ - tap	2. /e/ - egg	14. /ar/ - farm
3. /d/ - dog	15. /v/ - van	3. /i/ - in	15. /or/ - for
4. /f/ - fan	16. /w/ - wig	4. /o/ - on	16. /ur/ - hurt
5. /g/ - go	17. /y/ - yes	5. /u/ - up	17. /air/ - fair
6. /h/ - hen	18. /z/ - zip	6. /ai/ - rain	18. /ear/ - dear
7. /j/ - jet	19. /sh/ - shop	7. /ee/ - feet	19. /ure/[4] - sure
8. /l/ - leg	20. /ch/ - chip	8. /igh/ - night	20. /ə/ - corner (the 'schwa' - an unstressed vowel sound which is close to /u/)
9. /m/ - map	21. /th/ - thin	9. /oa/ - boat	
10. /n/ - net	22. /**th**/ - then	10. /**oo**/ - boot	
11. /p/ - pen	23. /ng/ - ring	11. /oo/ - look	
12. /r/ - rat	24. /zh/[3] - vision	12. /ow/ - cow	

Figure 1-4. Phonemes and examples

Morphemes and lexemes

A morpheme is the smallest unit of language that has a meaning. It is formed by a combination of phonemes. Not all morphemes are words, but all prefixes and suffixes are morphemes. For example, in the word "multimedia," "multi-" is not a word but a prefix that changes the meaning when put together with "media." "Multi-" is a morpheme. Figure 1-5 illustrates some words and their morphemes. For words like "cats" and "unbreakable," their morphemes are just constituents of the full word, whereas for words like "tumbling" and "unreliability," there is some variation when breaking the words down into their morphemes.

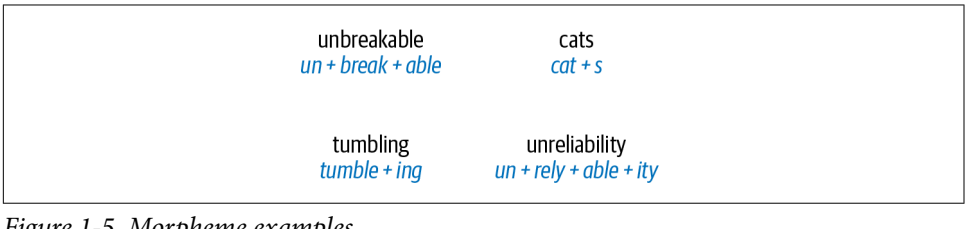

Figure 1-5. Morpheme examples

Lexemes are the structural variations of morphemes related to one another by meaning. For example, "run" and "running" belong to the same lexeme form. Morphological analysis, which analyzes the structure of words by studying its morphemes and lexemes, is a foundational block for many NLP tasks, such as tokenization, stemming,

learning word embeddings, and part-of-speech tagging, which we'll introduce in the next chapter.

Syntax

Syntax is a set of rules to construct grammatically correct sentences out of words and phrases in a language. Syntactic structure in linguistics is represented in many different ways. A common approach to representing sentences is a parse tree. Figure 1-6 shows an example parse tree for two English sentences.

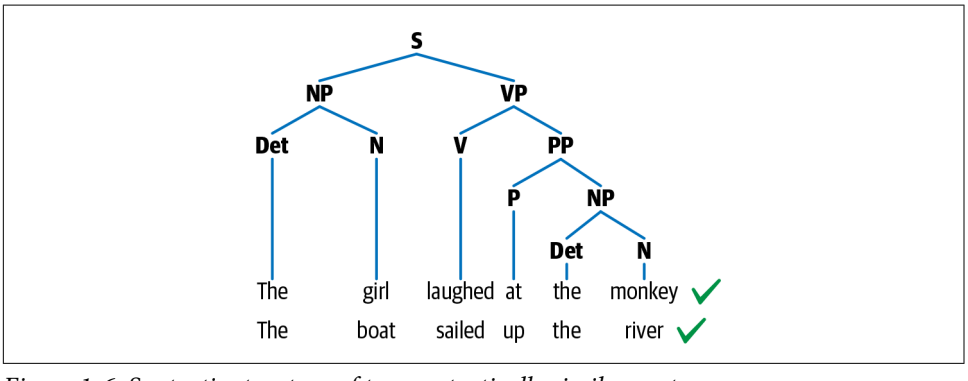

Figure 1-6. Syntactic structure of two syntactically similar sentences

This has a hierarchical structure of language, with words at the lowest level, followed by part-of-speech tags, followed by phrases, and ending with a sentence at the highest level. In Figure 1-6, both sentences have a similar structure and hence a similar syntactic parse tree. In this representation, N stands for noun, V for verb, and P for preposition. Noun phrase is denoted by NP and verb phrase by VP. The two noun phrases are "The girl" and "The boat," while the two verb phrases are "laughed at the monkey" and "sailed up the river." The syntactic structure is guided by a set of grammar rules for the language (e.g., the sentence comprises an NP and a VP), and this in turn guides some of the fundamental tasks of language processing, such as parsing. Parsing is the NLP task of constructing such trees automatically. Entity extraction and relation extraction are some of the NLP tasks that build on this knowledge of parsing, which we'll discuss in more detail in Chapter 5. Note that the parse structure described above is specific to English. The syntax of one language can be very different from that of another language, and the language-processing approaches needed for that language will change accordingly.

Context

Context is how various parts in a language come together to convey a particular meaning. Context includes long-term references, world knowledge, and common sense along with the literal meaning of words and phrases. The meaning of a sentence can change based on the context, as words and phrases can sometimes have multiple meanings. Generally, context is composed from semantics and pragmatics. Semantics is the direct meaning of the words and sentences without external context. Pragmatics adds world knowledge and external context of the conversation to enable us to infer implied meaning. Complex NLP tasks such as sarcasm detection, summarization, and topic modeling are some of tasks that use context heavily.

Linguistics is the study of language and hence is a vast area in itself, and we only introduced some basic ideas to illustrate the role of linguistic knowledge in NLP. Different tasks in NLP require varying degrees of knowledge about these building blocks of language. An interested reader can refer to the books written by Emily Bender [3, 4] on the linguistic fundamentals for NLP for further study. Now that we have some idea of what the building blocks of language are, let's see why language can be hard for computers to understand and what makes NLP challenging.

Why Is NLP Challenging?

What makes NLP a challenging problem domain? The ambiguity and creativity of human language are just two of the characteristics that make NLP a demanding area to work in. This section explores each characteristic in more detail, starting with ambiguity of language.

Ambiguity

Ambiguity means uncertainty of meaning. Most human languages are inherently ambiguous. Consider the following sentence: "I made her duck." This sentence has multiple meanings. The first one is: I cooked a duck for her. The second meaning is: I made her bend down to avoid an object. (There are other possible meanings, too; we'll leave them for the reader to think of.) Here, the ambiguity comes from the use of the word "made." Which of the two meanings applies depends on the context in which the sentence appears. If the sentence appears in a story about a mother and a child, then the first meaning will probably apply. But if the sentence appears in a book about sports, then the second meaning will likely apply. The example we saw is a direct sentence.

When it comes to figurative language—i.e., idioms—the ambiguity only increases. For example, "He is as good as John Doe." Try to answer, "How good is he?" The answer depends on how good John Doe is. Figure 1-7 shows some examples illustrating ambiguity in language.

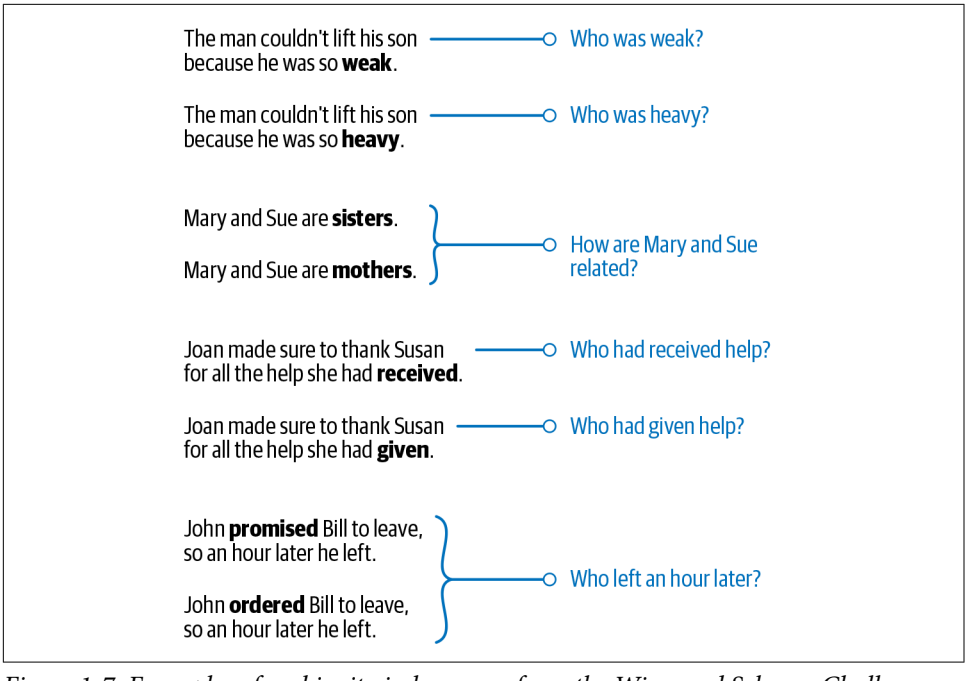

Figure 1-7. Examples of ambiguity in language from the Winograd Schema Challenge

The examples come from the Winograd Schema Challenge [5], named after Professor Terry Winograd of Stanford University. This schema has pairs of sentences that differ by only a few words, but the meaning of the sentences is often flipped because of this minor change. These examples are easily disambiguated by a human but are not solvable using most NLP techniques. Consider the pairs of sentences in the figure and the questions associated with them. With some thought, how the answer changes should be apparent based on a single word variation. As another experiment, consider taking an off-the-shelf NLP system like Google Translate and try various examples to see how such ambiguities affect (or don't affect) the output of the system.

Common knowledge

A key aspect of any human language is "common knowledge." It is the set of all facts that most humans are aware of. In any conversation, it is assumed that these facts are known, hence they're not explicitly mentioned, but they do have a bearing on the meaning of the sentence. For example, consider two sentences: "man bit dog" and "dog bit man." We all know that the first sentence is unlikely to happen, while the second one is very possible. Why do we say so? Because we all "know" that it is very unlikely that a human will bite a dog. Further, dogs are known to bite humans. This knowledge is required for us to say that the first sentence is unlikely to happen while the second one is possible. Note that this common knowledge was not mentioned in

either sentence. Humans use common knowledge all the time to understand and process any language. In the above example, the two sentences are syntactically very similar, but a computer would find it very difficult to differentiate between the two, as it lacks the common knowledge humans have. One of the key challenges in NLP is how to encode all the things that are common knowledge to humans in a computational model.

Creativity

Language is not just rule driven; there is also a creative aspect to it. Various styles, dialects, genres, and variations are used in any language. Poems are a great example of creativity in language. Making machines understand creativity is a hard problem not just in NLP, but in AI in general.

Diversity across languages

For most languages in the world, there is no direct mapping between the vocabularies of any two languages. This makes porting an NLP solution from one language to another hard. A solution that works for one language might not work at all for another language. This means that one either builds a solution that is language agnostic or that one needs to build separate solutions for each language. While the first one is conceptually very hard, the other is laborious and time intensive.

All these issues make NLP a challenging—yet rewarding—domain to work in. Before looking into how some of these challenges are tackled in NLP, we should know the common approaches to solving NLP problems. Let's start with an overview of how machine learning and deep learning are connected to NLP before delving deeper into different approaches to NLP.

Machine Learning, Deep Learning, and NLP: An Overview

Loosely speaking, artificial intelligence (AI) is a branch of computer science that aims to build systems that can perform tasks that require human intelligence. This is sometimes also called "machine intelligence." The foundations of AI were laid in the 1950s at a workshop organized at Dartmouth College [6]. Initial AI was largely built out of logic-, heuristics-, and rule-based systems. Machine learning (ML) is a branch of AI that deals with the development of algorithms that can learn to perform tasks automatically based on a large number of examples, without requiring handcrafted rules. Deep learning (DL) refers to the branch of machine learning that is based on artificial neural network architectures. ML, DL, and NLP are all subfields within AI, and the relationship between them is depicted in Figure 1-8.

While there is some overlap between NLP, ML, and DL, they are also quite different areas of study, as the figure illustrates. Like other early work in AI, early NLP applications were also based on rules and heuristics. In the past few decades, though, NLP

application development has been heavily influenced by methods from ML. More recently, DL has also been frequently used to build NLP applications. Considering this, let's do a short overview of ML and DL in this section.

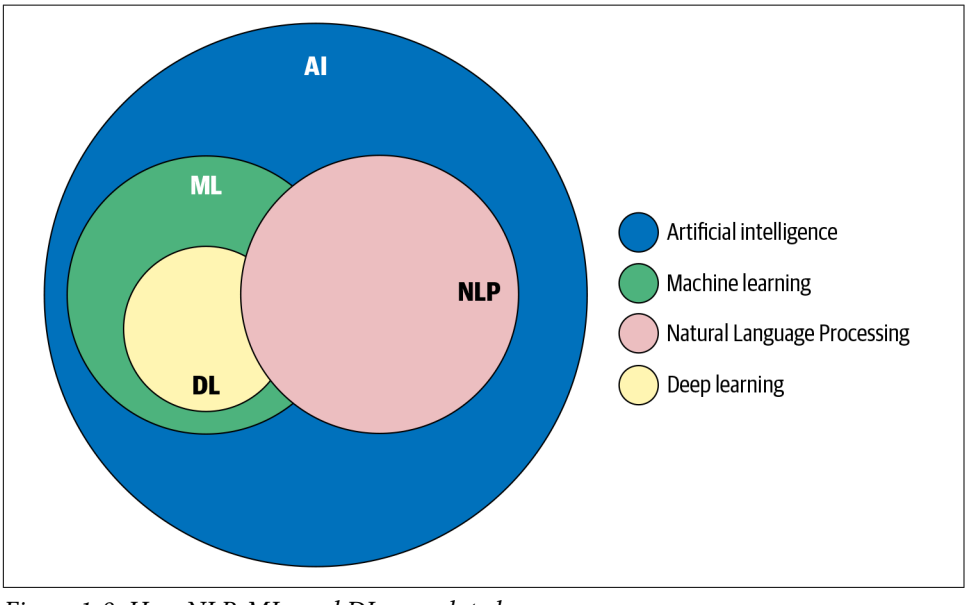

Figure 1-8. How NLP, ML, and DL are related

The goal of ML is to "learn" to perform tasks based on examples (called "training data") without explicit instruction. This is typically done by creating a numeric representation (called "features") of the training data and using this representation to learn the patterns in those examples. Machine learning algorithms can be grouped into three primary paradigms: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, the goal is to learn the mapping function from input to output given a large number of examples in the form of input-output pairs. The input-output pairs are known as *training data*, and the outputs are specifically known as *labels* or *ground truth*. An example of a supervised learning problem related to language is learning to classify email messages as spam or non-spam given thousands of examples in both categories. This is a common scenario in NLP, and we'll see examples of supervised learning throughout the book, especially in Chapter 4.

Unsupervised learning refers to a set of machine learning methods that aim to find hidden patterns in given input data without any reference output. That is, in contrast to supervised learning, unsupervised learning works with large collections of unlabeled data. In NLP, an example of such a task is to identify latent topics in a large collection of textual data without any knowledge of these topics. This is known as *topic modeling*, and we'll discuss it in Chapter 7.

Common in real-world NLP projects is a case of semi-supervised learning, where we have a small labeled dataset and a large unlabeled dataset. Semi-supervised techniques involve using both datasets to learn the task at hand. Last but not least, reinforcement learning deals with methods to learn tasks via trial and error and is characterized by the absence of either labeled or unlabeled data in large quantities. The learning is done in a self-contained environment and improves via feedback (reward or punishment) facilitated by the environment. This form of learning is not common in applied NLP (yet). It is more common in applications such as machine-playing games like go or chess, in the design of autonomous vehicles, and in robotics.

Deep learning refers to the branch of machine learning that is based on artificial neural network architectures. The ideas behind neural networks are inspired by neurons in the human brain and how they interact with one another. In the past decade, deep learning–based neural architectures have been used to successfully improve the performance of various intelligent applications, such as image and speech recognition and machine translation. This has resulted in a proliferation of deep learning–based solutions in industry, including in NLP applications.

Throughout this book, we'll discuss how all these approaches are used for developing various NLP applications. Let's now discuss the different approaches to solve any given NLP problem.

Approaches to NLP

The different approaches used to solve NLP problems commonly fall into three categories: heuristics, machine learning, and deep learning. This section is simply an introduction to each approach—don't worry if you can't quite grasp the concepts yet, as they'll be discussed in detail throughout the rest of the book. Let's jump in by discussing heuristics-based NLP.

Heuristics-Based NLP

Similar to other early AI systems, early attempts at designing NLP systems were based on building rules for the task at hand. This required that the developers had some expertise in the domain to formulate rules that could be incorporated into a program. Such systems also required resources like dictionaries and thesauruses, typically compiled and digitized over a period of time. An example of designing rules to solve an NLP problem using such resources is lexicon-based sentiment analysis. It uses counts of positive and negative words in the text to deduce the sentiment of the text. We'll cover this briefly in Chapter 4.

Besides dictionaries and thesauruses, more elaborate knowledge bases have been built to aid NLP in general and rule-based NLP in particular. One example is Wordnet [7], which is a database of words and the semantic relationships between them. Some

examples of such relationships are synonyms, hyponyms, and meronyms. Synonyms refer to words with similar meanings. Hyponyms capture is-type-of relationships. For example, baseball, sumo wrestling, and tennis are all hyponyms of sports. Meronyms capture is-part-of relationships. For example, hands and legs are meronyms of the body. All this information becomes useful when building rule-based systems around language. Figure 1-9 shows an example depiction of such relationships between words using Wordnet.

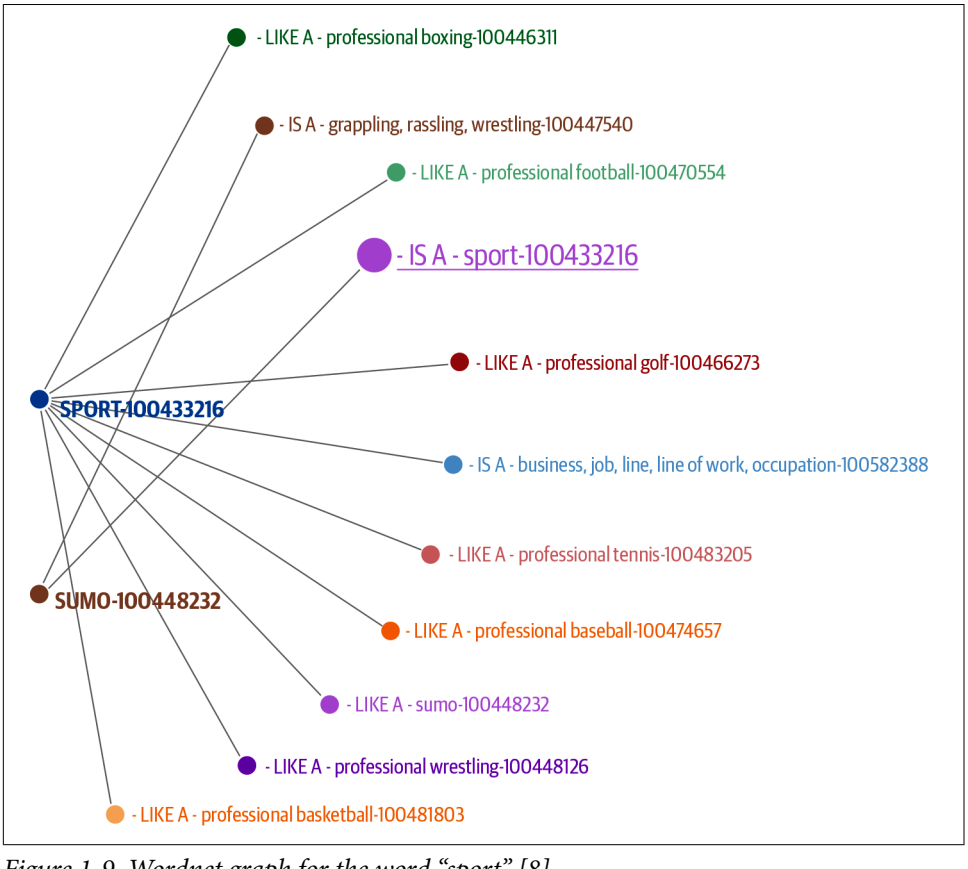

Figure 1-9. Wordnet graph for the word "sport" [8]

More recently, common sense world knowledge has also been incorporated into knowledge bases like Open Mind Common Sense [9], which also aids such rule-based systems. While what we've seen so far are largely lexical resources based on word-level information, rule-based systems go beyond words and can incorporate other forms of information, too. Some of them are introduced below.

Regular expressions (regex) are a great tool for text analysis and building rule-based systems. A regex is a set of characters or a pattern that is used to match and find

substrings in text. For example, a regex like '^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$' is used to find all email IDs in a piece of text. Regexes are a great way to incorporate domain knowledge in your NLP system. For example, given a customer complaint that comes via chat or email, we want to build a system to automatically identify the product the complaint is about. There is a range of product codes that map to certain brand names. We can use regexes to match these easily.

Regexes are a very popular paradigm for building rule-based systems. NLP software like StanfordCoreNLP includes TokensRegex [10], which is a framework for defining regular expressions. It is used to identify patterns in text and use matched text to create rules. Regexes are used for deterministic matches—meaning it's either a match or it's not. Probabilistic regexes is a sub-branch that addresses this limitation by including a probability of a match. Interested readers can look at software libraries such as pregex [11]. Last accessed June 15, 2020.

Context-free grammar (CFG) is a type of formal grammar that is used to model natural languages. CFG was invented by Professor Noam Chomsky, a renowned linguist and scientist. CFGs can be used to capture more complex and hierarchical information that a regex might not. The Earley parser [12] allows parsing of all kinds of CFGs. To model more complex rules, grammar languages like JAPE (Java Annotation Patterns Engine) can be used [13]. JAPE has features from both regexes as well as CFGs and can be used for rule-based NLP systems like GATE (General Architecture for Text Engineering) [14]. GATE is used for building text extraction for closed and well-defined domains where accuracy and completeness of coverage is more important. As an example, JAPE and GATE were used to extract information on pacemaker implantation procedures from clinical reports [15]. Figure 1-10 shows the GATE interface along with several types of information highlighted in the text as an example of a rule-based system.

Rules and heuristics play a role across the entire life cycle of NLP projects even now. At one end, they're a great way to build first versions of NLP systems. Put simply, rules and heuristics help you quickly build the first version of the model and get a better understanding of the problem at hand. We'll discuss this point in depth in Chapters 4 and 11. Rules and heuristics can also be useful as features for machine learning–based NLP systems. At the other end of the spectrum of the project life cycle, rules and heuristics are used to plug the gaps in the system. Any NLP system built using statistical, machine learning, or deep learning techniques will make mistakes. Some mistakes can be too expensive—for example, a healthcare system that looks into all the medical records of a patient and wrongly decides to not advise a critical test. This mistake could even cost a life. Rules and heuristics are a great way to plug such gaps in production systems. Now let's turn our attention to machine learning techniques used for NLP.

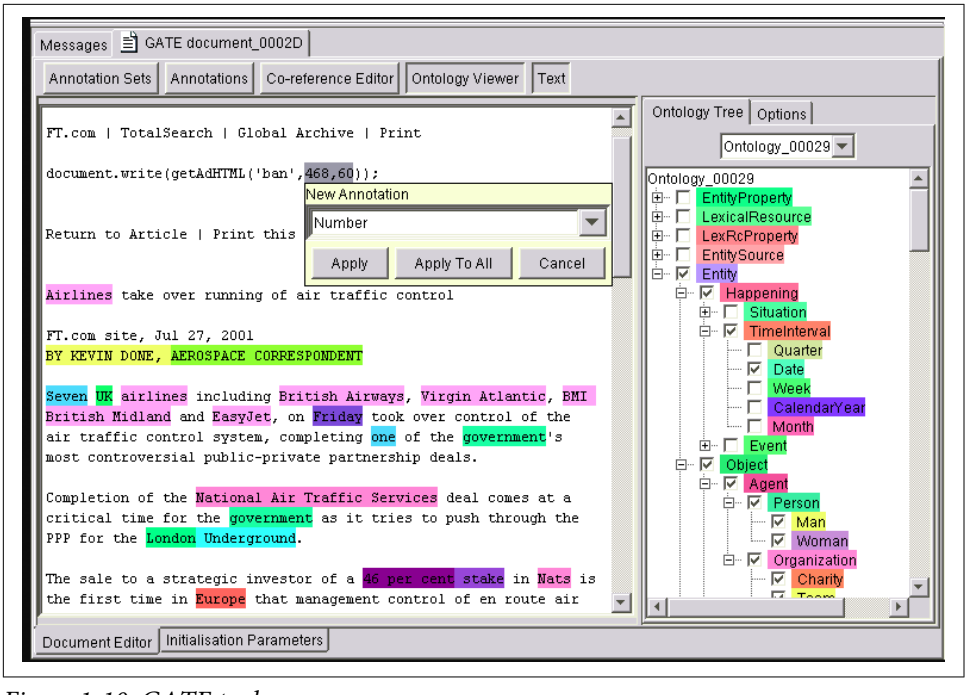

Figure 1-10. GATE tool

Machine Learning for NLP

Machine learning techniques are applied to textual data just as they're used on other forms of data, such as images, speech, and structured data. Supervised machine learning techniques such as classification and regression methods are heavily used for various NLP tasks. As an example, an NLP classification task would be to classify news articles into a set of news topics like sports or politics. On the other hand, regression techniques, which give a numeric prediction, can be used to estimate the price of a stock based on processing the social media discussion about that stock. Similarly, unsupervised clustering algorithms can be used to club together text documents.

Any machine learning approach for NLP, supervised or unsupervised, can be described as consisting of three common steps: extracting features from text, using the feature representation to learn a model, and evaluating and improving the model. We'll learn more about feature representations for text specifically in Chapter 3 and evaluation in Chapter 2. We'll now briefly outline some of the commonly used supervised ML methods in NLP for the second step (using the feature representation to learn a model). Having a basic idea of these methods will help you understand the concepts discussed in later chapters.

Naive Bayes

Naive Bayes is a classic algorithm for classification tasks [16] that mainly relies on Bayes' theorem (as is evident from the name). Using Bayes' theorem, it calculates the probability of observing a class label given the set of features for the input data. A characteristic of this algorithm is that it assumes each feature is independent of all other features. For the news classification example mentioned earlier in this chapter, one way to represent the text numerically is by using the count of domain-specific words, such as sport-specific or politics-specific words, present in the text. We assume that these word counts are not correlated to one another. If the assumption holds, we can use Naive Bayes to classify news articles. While this is a strong assumption to make in many cases, Naive Bayes is commonly used as a starting algorithm for text classification. This is primarily because it is simple to understand and very fast to train and run.

Support vector machine

The support vector machine (SVM) is another popular classification [17] algorithm. The goal in any classification approach is to learn a decision boundary that acts as a separation between different categories of text (e.g., politics versus sports in our news classification example). This decision boundary can be linear or nonlinear (e.g., a circle). An SVM can learn both a linear and nonlinear decision boundary to separate data points belonging to different classes. A linear decision boundary learns to represent the data in a way that the class differences become apparent. For two-dimensional feature representations, an illustrative example is given in Figure 1-11, where the black and white points belong to different classes (e.g., sports and politics news groups). An SVM learns an optimal decision boundary so that the distance between points across classes is at its maximum. The biggest strength of SVMs are their robustness to variation and noise in the data. A major weakness is the time taken to train and the inability to scale when there are large amounts of training data.

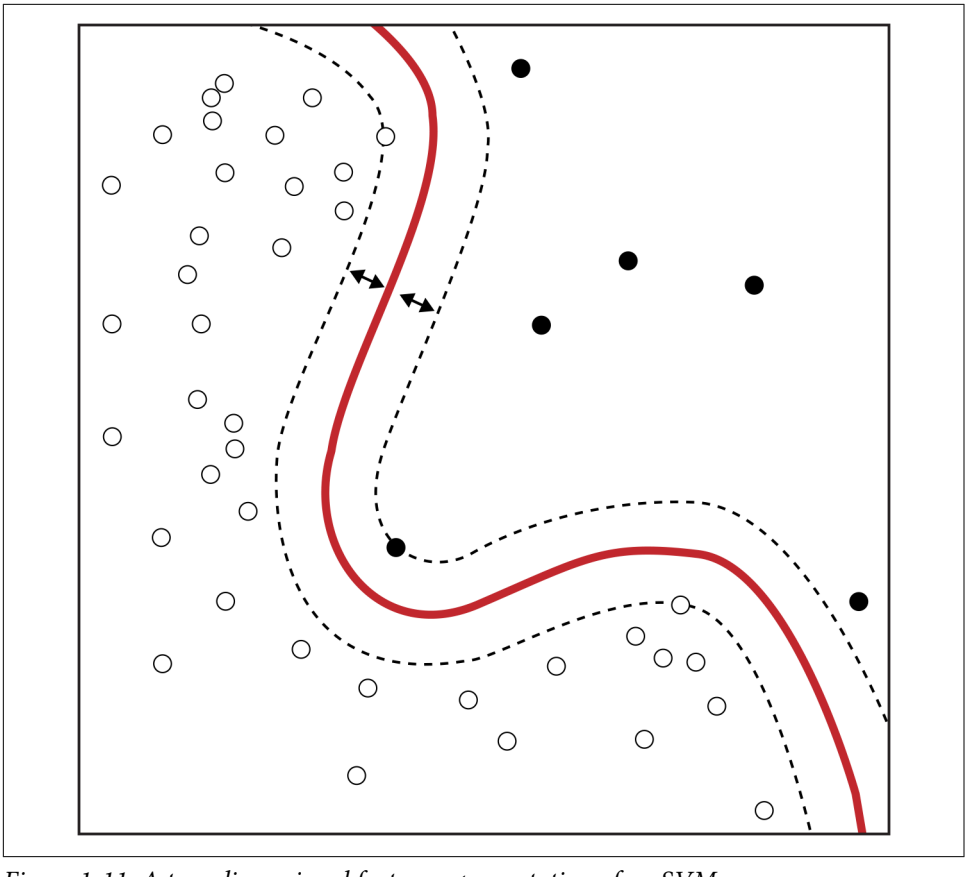

Figure 1-11. A two-dimensional feature representation of an SVM

Hidden Markov Model

The hidden Markov model (HMM) is a statistical model [18] that assumes there is an underlying, unobservable process with hidden states that generates the data—i.e., we can only observe the data once it is generated. An HMM then tries to model the hidden states from this data. For example, consider the NLP task of part-of-speech (POS) tagging, which deals with assigning part-of-speech tags to sentences. HMMs are used for POS tagging of text data. Here, we assume that the text is generated according to an underlying grammar, which is hidden underneath the text. The hidden states are parts of speech that inherently define the structure of the sentence following the language grammar, but we only observe the words that are governed by these latent states. Along with this, HMMs also make the Markov assumption, which means that each hidden state is dependent on the previous state(s). Human language is sequential in nature, and the current word in a sentence depends on what occurred before it. Hence, HMMs with these two assumptions are a powerful tool for modeling textual

data. In Figure 1-12, we can see an example of an HMM that learns parts of speech from a given sentence. Parts of speech like JJ (adjective) and NN (noun) are hidden states, while the sentence "natural language processing (nlp)…" is directly observed.

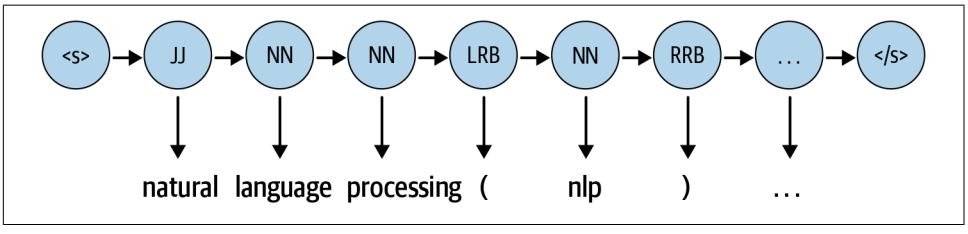

Figure 1-12. A graphical representation of a hidden Markov model

For a detailed discussion on HMMs for NLP, refer to Chapter 8 in the book *Speech and Language Processing* by Professor Jurafsky [19].

Conditional random fields

The conditional random field (CRF) is another algorithm that is used for sequential data. Conceptually, a CRF essentially performs a classification task on each element in the sequence [20]. Imagine the same example of POS tagging, where a CRF can tag word by word by classifying them to one of the parts of speech from the pool of all POS tags. Since it takes the sequential input and the context of tags into consideration, it becomes more expressive than the usual classification methods and generally performs better. CRFs outperform HMMs for tasks such as POS tagging, which rely on the sequential nature of language. We discuss CRFs and their variants along with applications in Chapters 5, 6, and 9.

These are some of the popular ML algorithms that are used heavily across NLP tasks. Having some understanding of these ML methods helps to understand various solutions discussed in the book. Apart from that, it is also important to understand when to use which algorithm, which we'll discuss in the upcoming chapters. To learn more about other steps and further theoretical details of the machine learning process, we recommend the textbook *Pattern Recognition and Machine Learning* by Christopher Bishop [21]. For a more applied machine learning perspective, Aurélien Géron's book [22] is a great resource to start with. Let's now take a look at deep learning approaches to NLP.

Deep Learning for NLP

We briefly touched on a couple of popular machine learning methods that are used heavily in various NLP tasks. In the last few years, we have seen a huge surge in using neural networks to deal with complex, unstructured data. Language is inherently complex and unstructured. Therefore, we need models with better representation and

learning capability to understand and solve language tasks. Here are a few popular deep neural network architectures that have become the status quo in NLP.

Recurrent neural networks

As we mentioned earlier, language is inherently sequential. A sentence in any language flows from one direction to another (e.g., English reads from left to right). Thus, a model that can progressively read an input text from one end to another can be very useful for language understanding. Recurrent neural networks (RNNs) are specially designed to keep such sequential processing and learning in mind. RNNs have neural units that are capable of remembering what they have processed so far. This memory is temporal, and the information is stored and updated with every time step as the RNN reads the next word in the input. Figure 1-13 shows an unrolled RNN and how it keeps track of the input at different time steps.

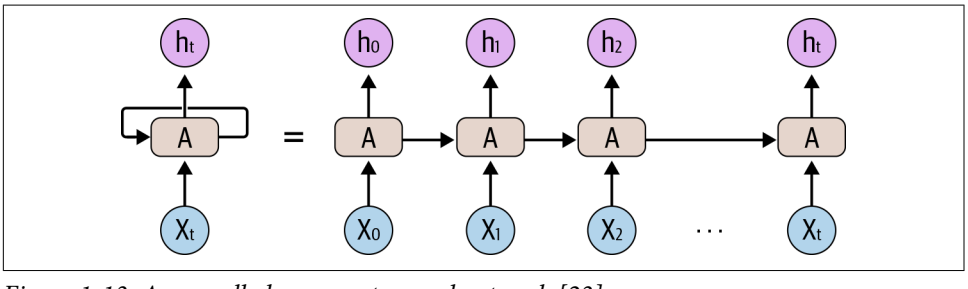

Figure 1-13. An unrolled recurrent neural network [23]

RNNs are powerful and work very well for solving a variety of NLP tasks, such as text classification, named entity recognition, machine translation, etc. One can also use RNNs to generate text where the goal is to read the preceding text and predict the next word or the next character. Refer to "The Unreasonable Effectiveness of Recurrent Neural Networks" [24] for a detailed discussion on the versatility of RNNs and the range of applications within and outside NLP for which they are useful.

Long short-term memory

Despite their capability and versatility, RNNs suffer from the problem of forgetful memory—they cannot remember longer contexts and therefore do not perform well when the input text is long, which is typically the case with text inputs. Long short-term memory networks (LSTMs), a type of RNN, were invented to mitigate this shortcoming of the RNNs. LSTMs circumvent this problem by letting go of the irrelevant context and only remembering the part of the context that is needed to solve the task at hand. This relieves the load of remembering very long context in one vector representation. LSTMs have replaced RNNs in most applications because of this workaround. Gated recurrent units (GRUs) are another variant of RNNs that are used mostly in language generation. (The article written by Christopher Olah [23] covers

the family of RNN models in great detail.) Figure 1-14 illustrates the architecture of a single LSTM cell. We'll discuss specific uses of LSTMs in various NLP applications in Chapters 4, 5, 6, and 9.

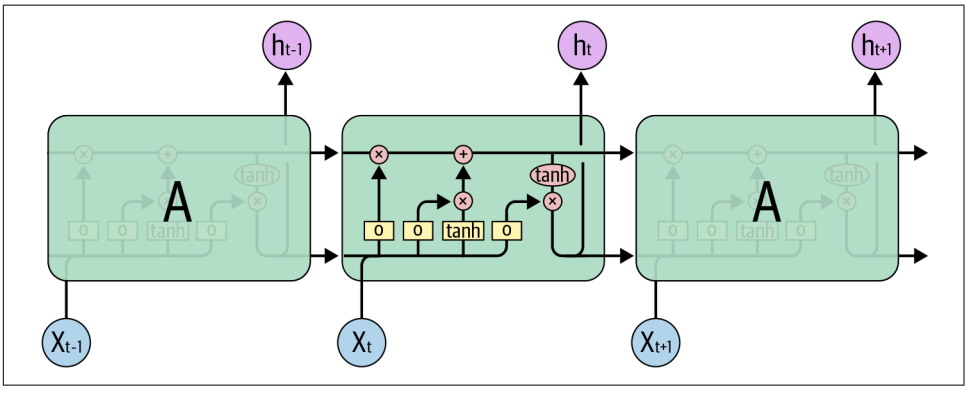

Figure 1-14. Architecture of an LSTM cell [23]

Convolutional neural networks

Convolutional neural networks (CNNs) are very popular and used heavily in computer vision tasks like image classification, video recognition, etc. CNNs have also seen success in NLP, especially in text-classification tasks. One can replace each word in a sentence with its corresponding word vector, and all vectors are of the same size (d) (refer to "Word Embeddings" in Chapter 3). Thus, they can be stacked one over another to form a matrix or 2D array of dimension $n \times d$, where n is the number of words in the sentence and d is the size of the word vectors. This matrix can now be treated similar to an image and can be modeled by a CNN. The main advantage CNNs have is their ability to look at a group of words together using a context window. For example, we are doing sentiment classification, and we get a sentence like, "I like this movie very much!" In order to make sense of this sentence, it is better to look at words and different sets of contiguous words. CNNs can do exactly this by definition of their architecture. We'll touch on this in more detail in later chapters. Figure 1-15 shows a CNN in action on a piece of text to extract useful phrases to ultimately arrive at a binary number indicating the sentiment of the sentence from a given piece of text.

As shown in the figure, CNN uses a collection of convolution and pooling layers to achieve this condensed representation of the text, which is then fed as input to a fully connected layer to learn some NLP tasks like text classification. More details on the usage CNNs for NLP can be found in [25] and [26]. We also cover them in Chapter 4.

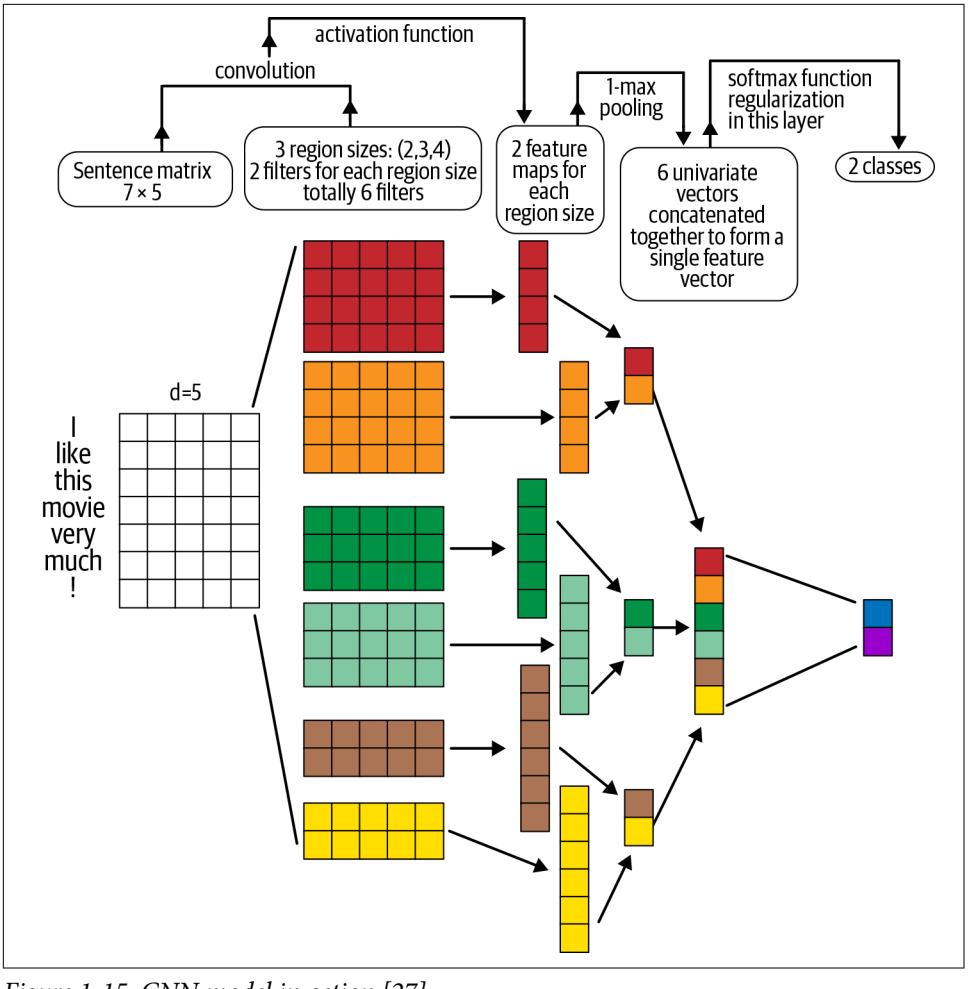

Figure 1-15. CNN model in action [27]

Transformers

Transformers [28] are the latest entry in the league of deep learning models for NLP. Transformer models have achieved state of the art in almost all major NLP tasks in the past two years. They model the textual context but not in a sequential manner. Given a word in the input, it prefers to look at all the words around it (known as *self-attention*) and represent each word with respect to its context. For example, the word "bank" can have different meanings depending on the context in which it appears. If the context talks about finance, then "bank" probably denotes a financial institution. On the other hand, if the context mentions a river, then it probably indicates a bank of the river. Transformers can model such context and hence have been used heavily

in NLP tasks due to this higher representation capacity as compared to other deep networks.

Recently, large transformers have been used for *transfer learning* with smaller downstream tasks. Transfer learning is a technique in AI where the knowledge gained while solving one problem is applied to a different but related problem. With transformers, the idea is to train a very large transformer mode in an unsupervised manner (known as *pre-training*) to predict a part of a sentence given the rest of the content so that it can encode the high-level nuances of the language in it. These models are trained on more than 40 GB of textual data, scraped from the whole internet. An example of a large transformer is BERT (Bidirectional Encoder Representations from Transformers) [29], shown in Figure 1-16, which is pre-trained on massive data and open sourced by Google.

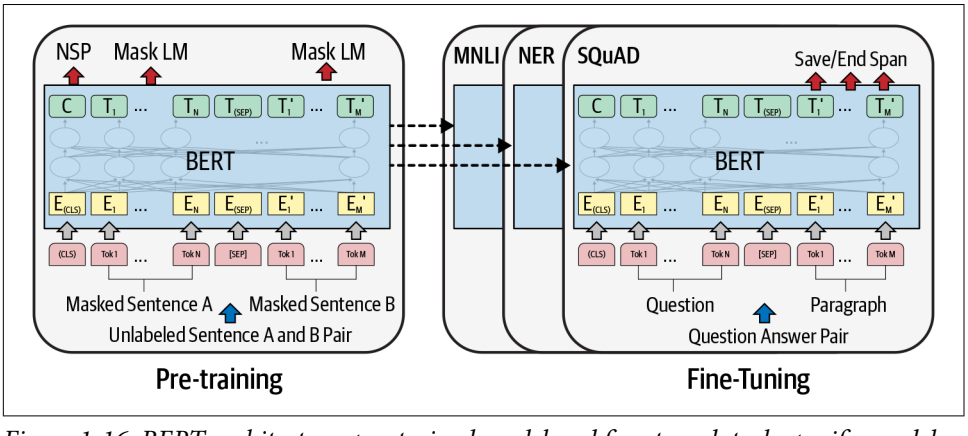

Figure 1-16. BERT architecture: pre-trained model and fine-tuned, task-specific models

The pre-trained model is shown on the left side of Figure 1-16. This model is then fine-tuned on downstream NLP tasks, such as text classification, entity extraction, question answering, etc., as shown on the right of Figure 1-16. Due to the sheer amount of pre-trained knowledge, BERT works efficiently in transferring the knowledge for downstream tasks and achieves state of the art for many of these tasks. Throughout the book, we have covered various examples of using BERT for various tasks. Figure 1-17 illustrates the workings of a self-attention mechanism, which is a key component of a transformer. Interested readers can look at [30] for more details on self-attention mechanisms and transformer architecture. We cover BERT and its applications in Chapters 4, 6, and 10.

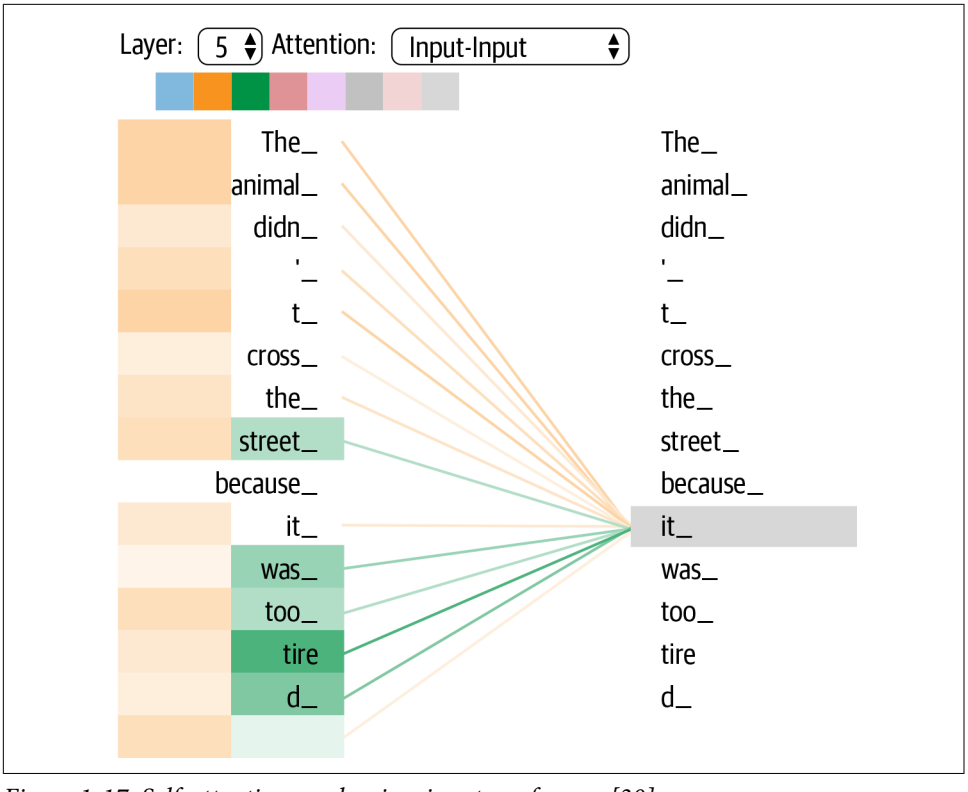

Figure 1-17. Self-attention mechanism in a transformer [30]

Autoencoders

An autoencoder is a different kind of network that is used mainly for learning compressed vector representation of the input. For example, if we want to represent a text by a vector, what is a good way to do it? We can learn a mapping function from input text to the vector. To make this mapping function useful, we "reconstruct" the input back from the vector representation. This is a form of unsupervised learning since you don't need human-annotated labels for it. After the training, we collect the vector representation, which serves as an encoding of the input text as a dense vector. Autoencoders are typically used to create feature representations needed for any downstream tasks. Figure 1-18 depicts the architecture of an autoencoder.

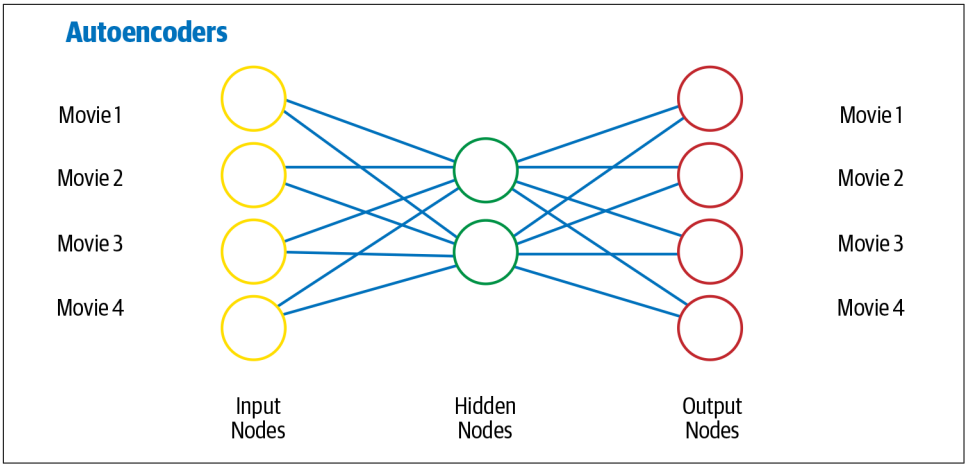

Figure 1-18. Architecture of an autoencoder

In this scheme, the hidden layer gives a compressed representation of input data, capturing the essence, and the output layer (decoder) reconstructs the input representation from the compressed representation. While the architecture of the autoencoder shown in Figure 1-18 cannot handle specific properties of sequential data like text, variations of autoencoders, such as LSTM autoencoders, address these well. More information about autoencoders can be found in [31].

We briefly introduced some of the popular DL architectures for NLP here. For a more detailed study of deep learning architectures in general, refer to [31], and specifically for NLP, refer to [25]. We hope this introduction gives you enough background to understand the use of DL in the rest of this book.

Going by all the recent achievements of DL models, one might think that DL should be the go-to way to build NLP systems. However, that is far from the truth for most industry use cases. Let's look at why this is the case.

Why Deep Learning Is Not Yet the Silver Bullet for NLP

Over the last few years, DL has made amazing advances in NLP. For example, in text classification, LSTM- and CNN-based models have surpassed the performance of standard machine learning techniques such as Naive Bayes and SVM for many classification tasks. Similarly, LSTMs have performed better in sequence-labeling tasks like entity extraction as compared to CRF models. Recently, powerful transformer models have become state of the art in most of these NLP tasks, ranging from classification to sequence labeling. A huge trend right now is to leverage large (in terms of number of parameters) transformer models, train them on huge datasets for generic NLP tasks like language models, then adapt them to smaller downstream tasks. This approach

(known as *transfer learning*) has also been successful in other domains, such as computer vision and speech.

Despite such tremendous success, DL is still not the silver bullet for all NLP tasks when it comes to industrial applications. Some of the key reasons for this are as follows:

Overfitting on small datasets
: DL models tend to have more parameters than traditional ML models, which means they possess more expressivity. This also comes with a curse. Occam's razor [32] suggests that a simpler solution is always preferable given that all other conditions are equal. Many times, in the development phase, sufficient training data is not available to train a complex network. In such cases, a simpler model should be preferred over a DL model. DL models overfit on small datasets and subsequently lead to poor generalization capability, which in turn leads to poor performance in production.

Few-shot learning and synthetic data generation
: In disciplines like computer vision, DL has made significant strides in few-shot learning (i.e., learning from very few training examples) [33] and in models that can generate superior-quality images [34]. Both of these advances have made training DL-based vision models on small amounts of data feasible. Therefore, DL has achieved much wider adoption for solving problems in industrial settings. We have not yet seen similar DL techniques be successfully developed for NLP.

Domain adaptation
: If we utilize a large DL model that is trained on datasets originating from some common domains (e.g., news articles) and apply the trained model to a newer domain that is different from the common domains (e.g., social media posts), it may yield poor performance. This loss in generalization performance indicates that DL models are not always useful. For example, models trained on internet texts and product reviews will not work well when applied to domains such as law, social media, or healthcare, where both the syntactic and semantic structure of the language is specific to the domain. We need specialized models to encode the domain knowledge, which could be as simple as domain-specific, rule-based models.

Interpretable models
: Apart from efficient domain adaptation, controllability and interpretability is hard for DL models because, most of the time, they work like a black box. Businesses often demand more interpretable results that can be explained to the customer or end user. In those cases, traditional techniques might be more useful. For example, a Naive Bayes model for sentiment classification may explain the effect of strong positive and negative words on the final prediction of sentiment.

As of today, obtaining such insights from an LSTM-based classification model is difficult. This is in contrast to computer vision, where DL models are not black boxes. There are plenty of techniques [35] in computer vision that are used to gain insight into why a model is making a particular prediction. Such approaches for NLP are not as common.

Common sense and world knowledge

Even though we have achieved good performance on benchmark NLP tasks using ML and DL models, language remains a bigger enigma to scientists. Beyond syntax and semantics, language encompasses knowledge of the world around us. Language for communication relies on logical reasoning and common sense regarding events from the world. For example, "I like pizza" implies "I feel happy when I eat pizza." A more complex reasoning example would be, "If John walks out of the bedroom and goes to the garden, then John is not in the bedroom anymore, and his current location is the garden." This might seem trivial to us humans, but it requires multistep reasoning for a machine to identify events and understand their consequences. Since this world knowledge and common sense are inherent in language, understanding them is crucial for any DL model to perform well on various language tasks. Current DL models may perform well on standard benchmarks but are still not capable of common sense understanding and logical reasoning. There are some efforts to collect common sense events and logical rules (such as if-them reasoning), but they are not well integrated yet with ML or DL models.

Cost

Building DL-based solutions for NLP tasks can be pretty expensive. The cost, in terms of both money and time, stems from multiple sources. DL models are known to be data guzzlers. Collecting a large dataset and getting it labeled can be very expensive. Owing to the size of DL models, training them to achieve desired performance can not only increase your development cycles but also result in a heavy bill for the specialized hardware (GPUs). Further, deploying and maintaining DL models can be expensive in terms of both hardware requirements and effort. Last but not least, because they're bulky, these models may cause latency issues during inference time and may not be useful in cases where low latency is a must. To this list, one can also add technical debt arising from building and maintaining a heavy model. Loosely speaking, technical debt is the cost of rework that arises from prioritizing speedy delivery over good design and implementation choices.

On-device deployment

For many use cases, the NLP solution needs to be deployed on an embedded device rather than in the cloud—for example, a machine-translation system that helps tourists speak the translated text even without the internet. In such cases, owing to limitations of the device, the solution must work with limited memory

and power. Most DL solutions do not fit such constraints. There are some efforts in this direction [36, 37, 38] where one can deploy DL models on edge devices, but we're still quite far from generic solutions.

In most industry projects, one or more of the points mentioned above plays out. This leads to longer project cycles and higher costs (hardware, manpower), and yet the performance is either comparable or sometimes even lower than ML models. This results in a poor return on investment and often causes the NLP project to fail.

Based on this discussion, it may be apparent that DL is not always the go-to solution for all industrial NLP applications. So, this book starts with fundamental aspects of various NLP tasks and how we can solve them using techniques ranging from rule-based systems to DL models. We emphasize the data requirements and model-building pipeline, not just the technical details of individual models. Given the rapid advances in this area, we anticipate that newer DL models will come in the future to advance the state of the art but that the fundamentals of NLP tasks will not change substantially. This is why we'll discuss the basics of NLP and build on them to develop models of increasing complexity wherever possible, rather than directly jumping to the cutting edge.

Echoing Professor Zachary Lipton from Carnegie Mellon University and Professor Jacob Steinhardt from UC Berkeley [39], we also want to provide a word of caution about consuming a lot of scientific articles, research papers, and blogs on ML and NLP without context and proper training. Following a large volume of cutting-edge work may cause confusion and not-so-precise understanding. Many recent DL models are not interpretable enough to indicate the sources of empirical gains. Lipton and Steinhardt also recognize the possible conflation of technical terms and misuse of language in ML-related scientific articles, which often fail to provide any clear path to solving the problem at hand. Therefore, in this book, we carefully describe various technical concepts in the application of ML in NLP tasks via examples, code, and tips throughout the chapters.

So far, we've covered some foundational concepts related to language, NLP, ML, and DL. Before we wrap up Chapter 1, let's look at a case study to help get a better understanding of the various components of an NLP application.

An NLP Walkthrough: Conversational Agents

Voice-based conversational agents like Amazon Alexa and Apple Siri are some of the most ubiquitous applications of NLP, and they're the ones most people are already familiar with. Figure 1-19 shows the typical interaction model of a conversational agent.

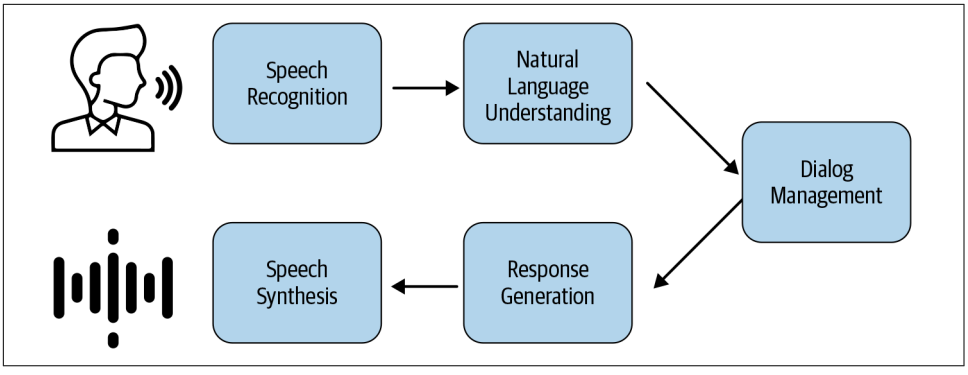

Figure 1-19. Flow of conversation agents

Here, we'll walk through all the major NLP components used in this flow:

1. *Speech recognition and synthesis*: These are the main components of any voice-based conversational agent. Speech recognition involves converting speech signals to their phonemes, which are then transcribed as words. Speech synthesis achieves the reverse process by transforming textual results into spoken language to the user. Both of these techniques have advanced considerably in the last decade, and we recommend using cloud APIs for most standard cases.
2. *Natural language understanding*: This is the next component in the conversational agent pipeline, where the user response received (transcribed as text) is analyzed using a natural language understanding system. This can be broken into many small NLP subtasks, such as:
   - *Sentiment analysis*: Here, we analyze the sentiment of the user response. This will be covered in Chapter 4.
   - *Named entity recognition*: Here, we identify all the important entities the user mentioned in their response. This will be covered in Chapter 5.
   - *Coreference resolution*: Here, we find out the references of the extracted entities from the conversation history. For example, a user may say "*Avengers Endgame* was awesome" and later refer back to the movie, saying "The movie's special effects were great." In this case, we would want to link that "movie" is referring to *Avengers Endgame*. This is covered briefly in Chapter 5.
3. *Dialog management*: Once we've extracted the useful information from the user's response, we may want to understand the user's intent—i.e., if they're asking a factual question like "What is the weather today?" or giving a command like "Play Mozart songs." We can use a text-classification system to classify the user response as one of the pre-defined intents. This helps the conversational agent know what's being asked. Intent classification will be covered in Chapters 4 and 6. During this process, the system may ask a few clarifying questions to elicit fur-

ther information from the user. Once we've figured out the user's intent, we want to figure out which suitable action the conversational agent should take to fulfill the user's request. This is done based on the information and intent extracted from the user's response. Examples of suitable actions could be generating an answer from the internet, playing music, dimming lights, or asking a clarifying question. We'll cover this in Chapter 6.

4. *Response generation*: Finally, the conversational agent generates a suitable action to perform based on a semantic interpretation of the user's intent and additional inputs from the dialogue with the user. As mentioned earlier, the agent can retrieve information from the knowledge base and generate responses using a pre-defined template. For example, it might respond by saying, "Now playing Symphony No. 25" or "The lights have been dimmed." In certain scenarios, it can also generate a completely new response.

We hope this brief case study provided an overview of how different NLP components we'll be discussing throughout this book will come together to build one application: a conversational agent. We'll see more details about these components as we progress through the book, and we'll discuss conversational agents specifically in Chapter 6.

Wrapping Up

From the broader contours of what a language is to a concrete case study of a real-world NLP application, we've covered a range of NLP topics in this chapter. We also discussed how NLP is applied in the real world, some of its challenges and different tasks, and the role of ML and DL in NLP. This chapter was meant to give you a baseline of knowledge that we'll build on throughout the book. The next two chapters (Chapters 2 and 3) will introduce you to some of the foundational steps necessary for building NLP applications. Chapters 4–7 focus on core NLP tasks along with industrial use cases that can be solved with them. In Chapters 8–10, we discuss how NLP is used across different industry verticals such as e-commerce, healthcare, finance, etc. Chapter 11 brings everything together and discusses what it takes to build end-to-end NLP applications in terms of design, development, testing, and deployment. With this broad overview in place, let's start delving deeper into the world of NLP.

References

[1] Arria.com. "NLG for Your Industry". Last accessed June 15, 2020.

[2] UCL. Phonetic symbols for English. Last accessed June 15, 2020.

[3] Bender, Emily M. "Linguistic Fundamentals for Natural Language Processing: 100 Essentials From Morphology and Syntax." *Synthesis Lectures on Human Language Technologies* 6.3 (2013): 1–184.

[4] Bender, Emily M. and Alex Lascarides. "Linguistic Fundamentals for Natural Language Processing II: 100 Essentials from Semantics and Pragmatics." *Synthesis Lectures on Human Language Technologies* 12.3 (2019): 1–268.

[5] Levesque, Hector, Ernest Davis, and Leora Morgenstern. "The Winograd Schema Challenge." *The Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning* (2012).

[6] Wikipedia. "Dartmouth workshop". Last modified March 30, 2020.

[7] Miller, George A. "WordNet: A Lexical Database for English." *Communications of the ACM* 38.11 (1995): 39–41.

[8] Visual Thesaurus of English Collocations. "Visual Wordnet with D3.js". Last accessed June 15, 2020.

[9] Singh, Push, Thomas Lin, Erik T. Mueller, Grace Lim, Travell Perkins, and Wan Li Zhu. "Open Mind Common Sense: Knowledge Acquisition from the General Public," Meersman R. and Tari Z. (eds), *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*. OTM 2002. *Lecture Notes in Computer Science*, vol. 2519. Berlin, Heidelberg: Springer.

[10] The Stanford Natural Language Processing Group. Stanford TokensRegex, (software). Last accessed June 15, 2020.

[11] Hewitt, Luke. Probabilistic regular expressions, (GitHub repo).

[12] Earley, Jay. "An Efficient Context-Free Parsing Algorithm." *Communications of the ACM* 13.2 (1970): 94–102.

[13] "Java Annotation Patterns Engine: Regular Expressions over Annotations". *Developing Language Processing Components with GATE Version 9 (a User Guide)*, Chapter 8. Last accessed June 15, 2020.

[14] General Architecture for Text Engineering (GATE). Last accessed June 15, 2020.

[15] Rosier, Arnaud, Anita Burgun, and Philippe Mabo. "Using Regular Expressions to Extract Information on Pacemaker Implantation Procedures from Clinical Reports." *AMIA Annual Symposium Proceedings* v.2008 (2008): 81–85.

[16] Zhang, Haiyi and Di Li. "Naïve Bayes Text Classifier." *2007 IEEE International Conference on Granular Computing* (GRC 2007): 708.

[17] Joachims, Thorsten. *Learning to Classify Text Using Support Vector Machines,* Vol. 668. New York: Springer Science & Business Media, 2002. ISBN: 978-1-4615-0907-3

[18] Baum, Leonard E. and Ted Petrie. "Statistical Inference for Probabilistic Functions of Finite State Markov Chains." *The Annals of Mathematical Statistics* 37.6 (1966): 1554–1563.

[19] Jurafsky, Dan and James H. Martin. *Speech and Language Processing*, Third Edition (Draft), 2018.

[20] Settles, Burr. "Biomedical Named Entity Recognition Using Conditional Random Fields and Rich Feature Sets." *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA/BioNLP)* (2004): 107–110.

[21] Bishop, Christopher M. *Pattern Recognition and Machine Learning*. New York: Springer, 2006. ISBN: 978-0-3873-1073-2

[22] Géron, Aurélien. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. Boston: O'Reilly, 2019. ISBN: 978-1-492-03264-9

[23] Olah, Christopher. "Understanding LSTM Networks". August 27, 2015.

[24] Karpathy, Andrej. "The Unreasonable Effectiveness of Recurrent Neural Networks". May 21, 2015.

[25] Goldberg, Yoav. "Neural Network Methods for Natural Language Processing." *Synthesis Lectures on Human Language Technologies* 10.1 (2017): 1–309.

[26] Britz, Denny. "Understanding Convolutional Neural Networks for NLP". November 7, 2015.

[27] Le, Hoa T., Christophe Cerisara, and Alexandre Denis. "Do Convolutional Networks need to be Deep for Text Classification?" *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[28] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need." *Advances in Neural Information Processing Systems*, 2017: 5998–6008.

[29] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". October 11, 2018.

[30] Alammar, Jay. "The Illustrated Transformer". June 27, 2018.

[31] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge: MIT Press, 2016. ISBN: 978-0-262-03561-3

[32] Varma, Nakul. COMS 4771: Introduction to Machine Learning, Lecture 6, Slide 7. Last accessed June 15, 2020.

[33] Wang, Yaqing, Quanming Yao, James Kwok, and Lionel M. Ni. "Generalizing from a Few Examples: A Survey on Few-Shot Learning", (2019).

[34] Wang, Zhengwei, Qi She, and Tomas E. Ward. "Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy", (2019).

[35] Olah, Chris, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. "The Building Blocks of Interpretability." *Distill* 3.3 (March 2018): e10.

[36] Nan, Kaiming, Sicong Liu, Junzhao Du, and Hui Liu. "Deep Model Compression for Mobile Platforms: A Survey." *Tsinghua Science and Technology* 24.6 (2019): 677–693.

[37] TensorFlow. "Get started with TensorFlow Lite". Last modified March 21, 2020.

[38] Ganesh, Prakhar, Yao Chen, Xin Lou, Mohammad Ali Khan, Yin Yang, Deming Chen, Marianne Winslett, Hassan Sajjad, and Preslav Nakov. "Compressing Large-Scale Transformer-Based Models: A Case Study on BERT", (2020).

[39] Lipton, Zachary C. and Jacob Steinhardt. "Troubling Trends in Machine Learning Scholarship", (2018).