Questions on Chapter 1

1. List examples of real world applications of NLP
2. Explain the following NLP tasks: language modelling, text classification, information extraction, information retrieval, conversational agent, text summarization, question answering, machine translation, and topic modelling.
3. What are the building blocks of language and their applications?
4. Why is NLP Challenging?
5. How NLP, ML, and DL are related?
6. Describe the heuristics-based NLP.
7. Explain briefly Naive Bayes, Support Vector Machine, Hidden Markov Model, And Conditional Random Fields approaches.
8. What is the difference between RNN and LSTM NN?
9. How CNN can be used for text processing?
10. Describe the concept transfer learning.
11. Give the architecture of autoencoder.
12. List the key reason that makes DL not suitable for all NLP tasks
13. Explain the flow of conversation agents.

--Answers--

1)

- 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.

2) NLP Tasks→

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.

3) building block:-

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).

4)Why Is 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.

دي زياده

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.

5)

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

6)

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

7)

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.

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

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.

8)

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.

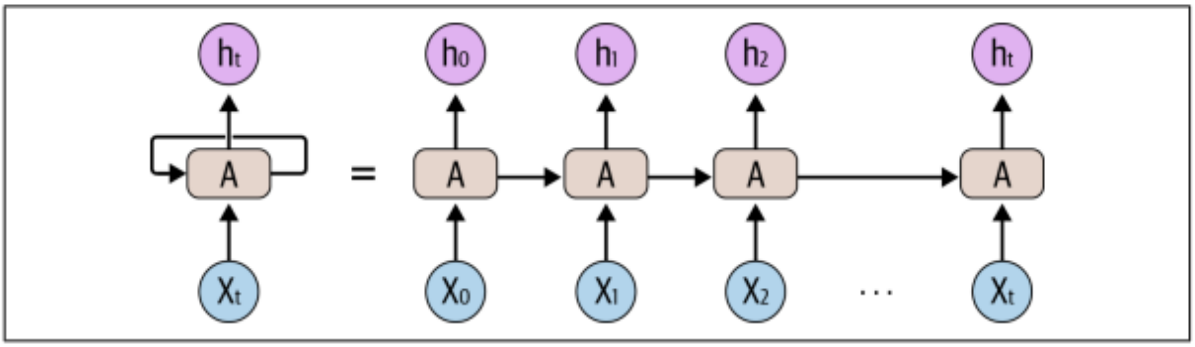

Figure 1-13. An unrolled recurrent neural network [23]

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

9)

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.

10)

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 trans-

دي زياده:

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

11)

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.

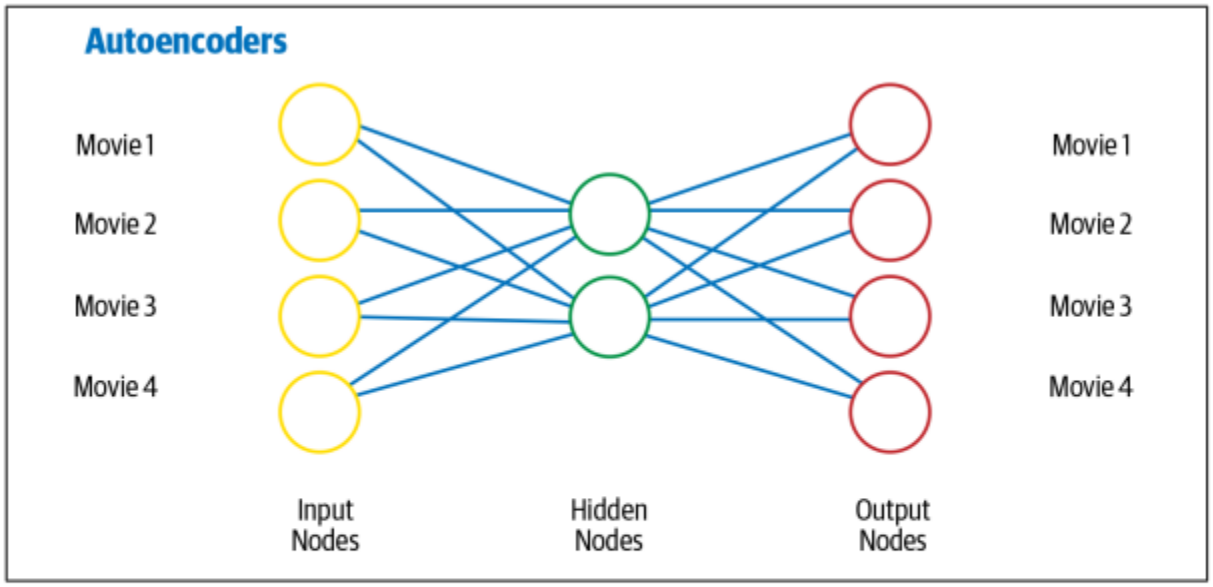

Figure 1-18. Architecture of an autoencoder

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,

12)

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

13)

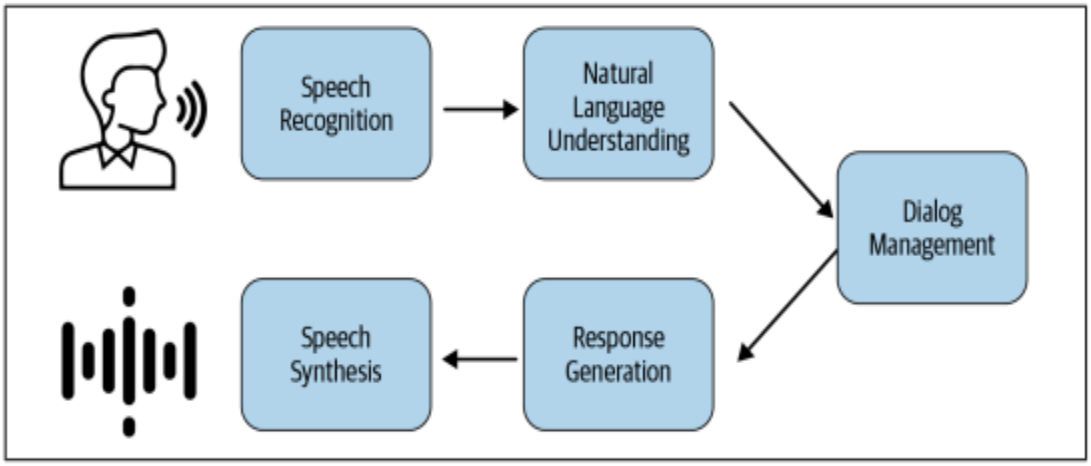

Figure 1-19. Flow of conversation agents

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.