Cognitive Computing and Big Data Analytics
By Judith Hurwitz, Marcia Kaufman and Adrian Bowles

CHAPTER
2

Design Principles for Cognitive Systems

In a cognitive computing system, the *model* refers to the corpus and the set of assumptions and algorithms that generate and score hypotheses to answer questions, solve problems, or discover new insights. How you model the world determines what kind of predictions you can make, patterns and anomalies you can detect, and actions you can take. The initial model is developed by the designers of the system, but the cognitive system will update the model and use the model to answer questions or provide insights. The *corpus* is the body of knowledge that machine learning algorithms use to continuously update that model based on its experience, which may include user feedback.

A cognitive system is designed to use a model of a domain to predict potential outcomes. Designing a cognitive system involves multiple steps. It requires an understanding of the available data, the types of questions that need to be asked, and the creation of a corpus comprehensive enough to support the generation of hypotheses about the domain based on observed facts. Therefore, a cognitive system is designed to create hypotheses from data, analyze alternative hypotheses, and determine the availability of supporting evidence to solve problems.

By leveraging machine learning algorithms, question analysis, and advanced analytics on relevant data, which may be structured or unstructured, a cognitive system can provide end users with a powerful approach to learning and decision making. Cognitive systems are designed to learn from their experiences

with data. A typical cognitive system uses machine learning algorithms to build models for answering questions or delivering insight. The design of a cognitive system needs to support the following differentiating characteristics:

- Access, manage, and analyze data in context.
- Generate and score multiple hypotheses based on the system's accumulated knowledge. A cognitive system may generate multiple possible solutions to every problem it solves and deliver answers and insights with associated confidence levels.
- The system continuously updates the model based on user interactions and new data. A cognitive system gets smarter over time in an automated way.

This chapter describes the major components that enable cognitive computing systems to learn, note the dependencies between these components, and outline the processes found within each component.

Components of a Cognitive System

A cognitive computing system has an internal store of knowledge (the corpus) and interacts with the external environment to capture additional data and to potentially update external systems. As discussed in Chapter 1, "The Foundation of Cognitive Computing," cognitive systems represent a new way of gaining insight from a diverse set of data resources. Cognitive systems may use Natural Language Processing to understand text, but also need other processing, deep learning capabilities, and tools to understand images, voice, videos, and location. These processing capabilities provide a way for the cognitive system to understand data in context and make sense of a particular domain area of knowledge. The cognitive system generates hypotheses and provides alternative answers or insights with associated confidence levels. In addition, a cognitive system needs to be capable of deep learning that is specific to subject areas and industries. The life cycle of a cognitive system is an iterative process. This iterative process requires the combination of human best practices and training of the data.

Figure 2-1 shows a representation of the typical elements in a cognitive computing system. This is intended as a general guide to cognitive computing architectures. In practice, cognitive components, APIs, and packaged services will emerge over time. However, even as services become embedded into systems, these elements will remain as the underpinning.

Now start your exploration of the design of a cognitive computing system by analyzing the corpus. Because the corpus is the knowledge base for the cognitive system, you are going to build a model of a specific domain by defining the corpus.

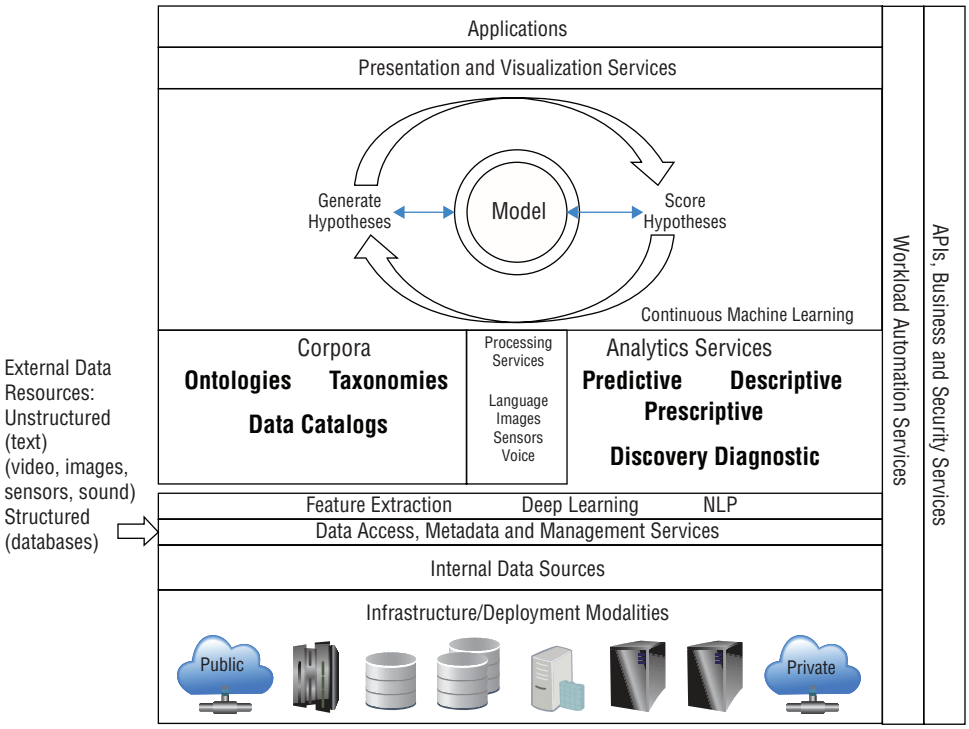

Figure 2-1: Architecture of a cognitive system

Building the Corpus

A *corpus* is a machine-readable representation of the complete record of a particular domain or topic. Experts in a variety of fields use a corpus or corpora for tasks such as linguistic analysis to study writing styles or even to determine authenticity of a particular work. For example, the complete works of William Shakespeare might be a corpus of interest to someone studying literature in the English Renaissance of the 16th and 17th century. For a researcher studying theatrical productions of the same period, it might be necessary to use the Shakespeare corpus plus several others that cover the works of his contemporaries. Such a collection of corpora can quickly become unwieldy if they come from different sources, have different formats, and particularly if they include a large volume of information that is irrelevant to the domain being studied. For example, someone studying theatrical productions might not be interested in the Shakespearean sonnets. Deciding what to leave out is as important as what to include.

In a cognitive computing application, the corpus or corpora represent the body of knowledge the system can use to answer questions, discover new patterns

or relationships, and deliver new insights. Before the system is launched, however, a base corpus must be created and the data ingested. The contents of this base corpus constrain the types of problems that can be solved, and the organization of data within the corpus has a significant impact on the efficiency of the system. Therefore, you need a good understanding of the domain area for your cognitive system before determining the required data sources. What types of problems do you want to solve? If the corpus is too narrowly defined, you may miss out on new and unexpected insights. If data is trimmed from the external sources before they are imported into the corpus, they won't be used in the generation or scoring of hypotheses, which is at the heart of machine learning. The corpus needs to include the right mix of relevant data resources that can enable the cognitive system to deliver accurate responses in the expected timeframe. When developing a cognitive system, it's a good idea to err on the side of gathering more data or knowledge because you never know when the discovery of an unexpected association will lead to important new knowledge.

Given the importance placed on having the right mix of data sources, a number of questions have to be addressed early in the design phase for a cognitive computing system:

- Which internal and external data sources are needed for the specific domain areas and problems to be solved? Will external data sources be ingested in whole or in part?
- How can you optimize the organization of data for efficient search and analysis?
- How can you integrate data across multiple corpora?
- How can you ensure that the corpus is expanded to fill in knowledge gaps in your base corpus? How can you determine which data sources need to be updated and at what frequency?

The choice of which sources to include in the initial corpus is critical. Sources ranging from medical journals to Wikipedia may now be efficiently imported in preparation for the launch of a cognitive system. In addition, it may be equally important to ingest information from videos, images, voice, and sensors. These sources are ingested at the data access layer (refer to Figure 2-1). Other data sources may also include subject-specific structured databases, ontologies, taxonomies, and catalogs.

If the cognitive computing application requires access to highly structured data created by or stored in other systems such as public or proprietary databases, another design consideration is how much of that data to import initially. It is also important to determine whether to update or refresh the data periodically, continuously, or in response to a request from the system when it recognizes that more data can help it provide better answers.

In many fields, taxonomies are used to capture hierarchical relationships between elements of interest. For example, a taxonomy for the U.S. Generally Accepted Accounting Principles (GAAP) represents the accounting standards in a hierarchical structure that capture the relationships between them. An *ontology* is similar to a taxonomy, but generally represents more complex relationships, such as the mapping between symptoms and diagnostic criteria in the Diagnostic and Statistical Manual of the American Psychiatric Association. When such a generally accepted taxonomy or ontology exists in a field, it may be useful to import that structure with its data—in whole or in part—rather than creating a novel structure for the same data. Chapter 5, "Representing Knowledge in Taxonomies and Ontologies," discusses the roles of taxonomies and ontologies in more detail.

During the design phase of a cognitive system, a key consideration is whether to construct a taxonomy or ontology if none already exist for the domain. Having such a structure may simplify the operation of the system and make it more efficient. However, if the designers are responsible for ensuring that a taxonomy or ontology is complete and up to date, it may be more effective to have the system continuously evaluate relationships between domain elements rather than have the designers build that into a hard-coded structure.

The choice of data structures can greatly impact the performance of the system on repetitive tasks such as knowledge retrieval for generating and scoring hypotheses. It is therefore advisable to model or simulate typical workloads during the design phase before committing to specific structures. A data catalog, which includes metadata such as semantic information or pointers, may be used to manage the underlying data more efficiently. The catalog is, as an abstraction, more compact and generally faster to manipulate than the much larger database it represents.

In the examples and diagrams, when referring to corpora, it should be noted that these can be integrated into a single corpus when doing so will help simplify the logic of the system or improves performance. Much like a system can be defined as a collection of smaller integrated systems, aggregating data from a collection of corpora results in a single new corpus. Maintaining separate corpora is typically done for performance reasons, much like normalizing tables in a database to facilitate queries, rather than attempting to combine tables into a single, more complex structure.

Corpus Management Regulatory and Security Considerations

Data sources and the movement of that data are increasingly becoming heavily regulated, particularly for personally identifiable information. Some general issues of data policies for protection, security, and compliance are common to all applications, but cognitive computing applications learn and derive new data or knowledge that may also be subject to a growing body of state, federal, and international legislation.

When the initial corpus is developed, it is likely that a lot of data will be imported using extract-transform-load (ETL) tools. These tools may have risk management, security, and regulatory features to help the user guard against data misuse or provide guidance when sources are known to contain sensitive data. The availability of these tools doesn't absolve the developers from responsibility to ensure that the data and metadata is in compliance with applicable rules and regulations. Protected data may be ingested (for example, personal identifiers) or generated (for example, medical diagnoses) when the corpus is updated by the cognitive computing system. Planning for good corpus management should include a plan to monitor relevant policies that impact data in the corpus. The data access layer tools described in the next section must be accompanied by or embed compliance policies and procedures to ensure that imported and derived data and metadata remain in compliance. That includes consideration of various deployment modalities, such as cloud computing, which may distribute data across geopolitical boundaries.

Bringing Data into the Cognitive System

Unlike many traditional systems, the data that is ingested into the corpus is not static. You need to build a base of knowledge that adequately defines your domain space. You begin populating this knowledge base with data you expect to be important. As you develop the model in the cognitive system, you refine the corpus. Therefore, you will continuously add to the data sources, transform those data sources, and refine and cleanse those sources based on the model development and continuous learning. This next section discusses how both data sources that are internal to an organization and those that are external to the organization can be used to build a corpus.

Leveraging Internal and External Data Sources

Most organizations already manage huge volumes of structured data from their transactional systems and business applications, and unstructured data such as text contained in forms or notes and possibly images from documents or corporate video sources. Although some firms are writing applications to monitor external sources such as news and social media feeds, many IT organizations are not yet well equipped to leverage these sources and integrate them with internal data sources. Most cognitive computing systems will be developed for domains that require ongoing access to integrated data from outside the organization.

Just as an individual learns to identify the right external sources to support decision making—from newspapers to network news to social media on the Internet—a cognitive computing system generally needs to access a variety

of frequently updated sources to keep current about the domain in which it operates. Also, like professionals who must balance the news or data from these external sources against their own experience, a cognitive system must learn to weigh the external evidence and develop confidence in the source as well as the content over time. For example, a popular magazine with articles on psychology may be a valuable resource, but if it contains data that is in conflict with a refereed journal article on the same topic, the system must know how to weigh the opposing positions. All data sources that may be useful should be considered and potentially ingested. However, this does not mean that all sources will be of equal value.

In healthcare, for example, electronic medical records (EMRs) can provide valuable source information. Although individuals are not always accurate in their own recollections, a database that aggregates findings from a broad spectrum of EMRs and case files may contain information about relationships between combinations of symptoms and disorders or diseases that would be missed if a doctor or researcher had access only to the records from their own practice or institution. In telecommunications, a company might want to use a cognitive system to anticipate machine failures based on internal factors, such as traffic and usage patterns, and on external factors such as severe weather threats that are likely to cause overloads and physical damage. You see more examples of integrating internal and external data sources in Chapter 12, "Smarter Cities: Cognitive Computing in Government." The important thing to remember at the design stage is that with experience, a cognitive computing system should identify and request additional data from external sources when that data will enable it to make better decisions or recommendations. Determining the right data at the right time to make a decision may always be problematic, but with a cognitive computing system, every request for more data will be based on an immediate need.

Data Access and Feature Extraction Services

The data access level of the diagram shown in Figure 2-1 depicts the main interfaces between the cognitive computing system and the outside world. Any data to be imported from external sources must come through processes within this layer. Cognitive computing applications may leverage external data sources in formats as varied as natural language text, video images, audio files, sensor data, and highly structured data formatted for machine processing. The analogy to human learning is that this level represents the senses. The feature extraction layer has to complete two tasks. First, it has to identify relevant data that needs to be analyzed. The second task is to abstract data as required to support machine learning.

The data access level is shown as separate but closely bound to the feature extraction level to reinforce the idea that some data must be captured and then

analyzed or refined before it is ready to be integrated into a corpus suitable for a particular domain. Any data that is considered unstructured—from video and images to natural language text—must be processed in this layer to find the underlying structure. Feature extraction and deep learning refer to a collection of techniques—primarily statistical algorithms—used to transform data into representations that capture the essential properties in a more abstract form that can be processed by a machine learning algorithm. For example, image data generally starts as a sparse binary representation that captures data about individual pixels but doesn't directly represent the underlying objects in the image. A digital image of a cat or a cat scan would be useless to a veterinary or a radiology cognitive computing system until the underlying structure was identified and represented in more meaningful way. Similarly, unstructured text becomes useful input to a cognitive system only when its meaning has been uncovered by a Natural Language Processing System (NLP is covered in detail in Chapter 3, "Natural Language Processing in Support of a Cognitive System").

Although these layers appear as a straightforward process of importing and refining data, it should be noted that external sources may be added or removed based on their value in hypothesis generation and scoring over time. For example, a medical diagnosis system may add a new external source of case files or delete a journal if it is found to provide unreliable evidence. For cognitive systems that provide evidence to support hypotheses in regulated industries, the data access layer processes or the corpora management services should maintain a log or other state data so that an auditor can determine what was "known" at any point in time. This would be important, for example, if a recommendation from the cognitive system was followed by a practitioner and resulted in harm (from a medical misdiagnosis to a poor recommendation by an accountant).

Analytics Services

Analytics refers to a collection of techniques used to find and report on essential characteristics or relationships within a data set. In general, the use of an analytic technique provides insights about the data to guide some action or decision. A number of packaged algorithms such regression analysis are widely used within solutions. Within a cognitive system, a wide range of standard analytics components are available for descriptive, predictive, and prescriptive tasks within statistical software packages or in commercial component libraries. A variety of tools that support various tasks within cognitive computing systems are available (refer to Figure 2-1). A cognitive computing system generally has additional analytical components embedded in the machine learning cycle algorithms. Chapter 6, "Applying Advanced Analytics to Cognitive Computing," goes into detail about the relevant analytical methods.

Machine Learning

Continuous learning without reprogramming is at the heart of all cognitive computing solutions. Although the techniques used to acquire, manage, and learn from data vary greatly, at their core most systems apply algorithms developed by researchers in the field of machine learning. Machine learning is a discipline that draws from computer science, statistics, and psychology.

Finding Patterns in Data

A typical machine-learning algorithm looks for patterns in data and then takes or recommends some action based on what it finds. A pattern may represent a similar structure (for example, elements of a picture that indicate a face), similar values (a cluster of values similar to those found in another data set) or proximity (how "close" the abstract representation of one item is to another). *Proximity* is an important concept in pattern identification or matching. Two data strings representing things or concepts in the real world are "close" when their abstract binary representations have similar characteristics.

Cognitive computing systems use machine-learning algorithms based on inferential statistics to detect or discover patterns that guide their behavior. Choosing the basic learning approach to adopt—detection versus discovery of patterns—should be based on the available data and nature of the problem to be solved. Machine learning typically uses *inferential statistics* (the basis for predictive, rather than descriptive analytics) techniques.

Next, look at two complementary approaches to machine learning that use patterns in different ways: supervised and unsupervised learning. Deciding when to use one or both of these approaches for a specific system depends on the attributes of available data and the goals of the system.

Finding the right machine learning algorithm or algorithms for a cognitive computing application starts with a few questions:

- Is there an existing source of data and associations between data elements to solve my problem?
- Do I know what kind of patterns my data contains?
- Can I give examples of how I would identify and exploit these patterns manually?

When all these questions can be answered in the affirmative, you have a good candidate for a supervised learning system.

Supervised Learning

Supervised learning refers to an approach that teaches the system to detect or match patterns in data based on examples it encounters during training with

sample data. The training data should include examples of the types of patterns or question-answer pairs the system will have to process. Learning by example, or *modeling*, is a powerful teaching technique that can be used for training systems to solve complex problems. After the system is operational, a supervised learning system also uses its own experience to improve its performance on pattern matching tasks.

Supervised learning can be used with data that is known to predict outcomes and results. Supervised learning requires an external system—the developer or user—that can evaluate or create a sample data set that represents the domain of data that the system will encounter when it is operational.

In supervised learning, the job of the algorithm is to create a mapping between input and output. The supervised learning model has to process enough data to get to the wanted level of validation, usually expressed as accuracy on the test data set. Both the training data and independent test data should be representative of the type of data that will be encountered when the system is operational. The starting point usually includes noisy data (data that includes a lot of extraneous details that are irrelevant) in the beginning that can be culled as it is trained. There needs to be enough training data so that it is possible to pinpoint the well-conceived hypothesis from the hypothesis class. Achieving this goal with supervised data requires good optimization methods to find that correct hypothesis from the training data. Biases and assumptions are always reflected in a training data set that may affect the performance of the system. It may be necessary to retrain a system when topics or responses drift from the model created by the original assumptions.

The primary applications for supervised learning are in systems that solve classification or regression problems. Solving these problems manually requires a person to recognize patterns based on experience or evidence, and identify one or more answers that satisfy all the constraints of the problem (classification problem) or fill in expected values (regression problem). Experienced people do this well, from a travel agent finding just the right vacation elements for a regular customer to a realtor predicting a selling price on a house. The appeal of supervised learning systems for cognitive computing is that they can operate on much larger data sets than humans can handle, so they are not only more efficient, they also can become more effective than humans with sufficient experience. The learning process begins with an established set of data and an understanding of how that data is organized, including the relationships between attributes of questions and answers. It can then proceed with inductive learning to generalize from the specific examples. Therefore, for supervised learning, the developer needs to begin with a model where the parameters are discoverable from the data set that is used for training.

In a classification system, the goal is to find a match between a set of objectives and a discrete solution. For example, in a cognitive travel application, it is necessary to learn about the desires of travelers and provide suggestions and recommendations for travel. The matching algorithm then compares these

features with the data from many other travel requests over time. The algorithm might take into account whether it is a return customer or someone logging on for the first time. Through a question-and-answer process, the system can begin to gain an understanding of who the customer is and what type of offerings would be of interest to the customer. Therefore, the system needs to identify the relevant attributes of that person so that it can begin to narrow down the set of possible responses.

Over time, the system adds more and more data from a large population of users. The learning process begins to see more patterns and builds the connections and context. The algorithms, for example, gain insights into relationships between different categories or clusters of travelers and their preferences. The more data sources that are included in the corpus, the better the learning system can be at suggesting options that will satisfy travelers.

In contrast with classification problems, regression problems require the system to determine the value of a continuous variable, such as price. The system must determine a value based on similar data with known answers. For example, using data about recent automobile sales in a geographic area, including make, model, mileage, and condition, a system can provide an estimate of a specific automobile's selling price by finding a near match with simple regression analysis. This is a typical predictive analytics problem. A supervised learning system can be taught to look for additional attributes that correlate with price and offer more precision as it learns from experience with more data.

Reinforcement Learning

Reinforcement learning is a special case of supervised learning in which the cognitive computing system receives feedback on its performance to guide it to a goal or good outcome. Unlike other supervised learning approaches, however, with reinforcement learning, the system is not explicitly trained with sample data. In reinforcement learning, the system learns to take next actions based on trial and error. Some typical applications of reinforcement learning include robotics and game playing. The machine learning algorithms assess the goodness or effectiveness of policies or actions, and reward the most effective actions. A sequence of successful decisions results in reinforcement, which helps the system generate a policy that best matches the problem being addressed.

Reinforcement learning for cognitive computing is most appropriate where a system must perform a sequence of tasks and the number of variables would make it too difficult to develop a representative training data set. For example, reinforcement would be used in robotics or a self-driving car. The learning algorithm must discover an association between the reward and a sequence of events leading up to the reward. Then the algorithm can try to optimize future actions to remain in a reward state. Although animals can be rewarded with

praise, food, or even cash, the reward function in a machine-learning environment is numerical or logical in nature.

Unsupervised Learning

Unsupervised learning refers to a machine learning approach that uses inferential statistical modeling algorithms to *discover* rather than *detect* patterns or similarities in data. An unsupervised learning system can identify new patterns, instead of trying to match a set of patterns it encountered during training. Unlike supervised learning, unsupervised learning is based solely on experience with the data rather than on training with sample data. Unsupervised learning requires the system to discover which relationships among data elements or structures are important by analyzing attributes like frequency of occurrence, context (for example, what has been seen or what has occurred previously), and proximity.

Unsupervised learning is the best approach for a cognitive computing system when an expert or user cannot give examples of typical relationships or question-answer pairs as guides to train the system. This may be due to the complexity of the data, when there are too many variables to consider, or when the structure of the data is unknown (for example, evaluating images from a surveillance camera to detect which person or persons are behaving differently from the crowd). The system first has to identify the actors, then their behaviors, and then find anomalies.

Unsupervised learning is also appropriate when new patterns emerge faster than humans can recognize them so that regular training is impossible. For example, a cognitive computing system to evaluate network threats must recognize anomalies that may indicate an attack or vulnerability that has never been seen before. By comparing the current state of the network with historical data, an unsupervised learning system can look for changes or a state it has never seen before and flag that as suspect activity. If the activity is benign, the system can learn from that experience not to flag that state if it sees it again in the future.

In essence, with unsupervised learning you begin with a massive amount of data, and without preconceived notions about the patterns, relationships, or associations that may be found. In unsupervised learning, you expect that the data will reveal the patterns and anomalies through statistical analysis. Therefore, the goal of unsupervised learning is to discover patterns in the data in the absence of an explicit training model. This type of learning requires sophisticated mathematics and often requires the use of clustering (gathering like data elements) and hidden Markov models (finding patterns that appear over a space of time, such as speech). Unsupervised learning is typically used in areas such as vision analysis, imaging, and bioinformatics for gene or protein sequencing.

Unlike supervised learning, there is no distinction between training and test data, and there is no specific training data that incorporates the patterns being sought in fresh test data.

USING UNSUPERVISED DISCOVERIES TO DRIVE SUPERVISED LEARNING

For some domains, a hybrid approach that uses both supervised and unsupervised learning components will be the most effective approach. When an unsupervised learning system detects interesting patterns, knowledge about the associations and relationships within the patterns may be used to construct training data for a supervised learning system. For example, the retail system mentioned that could detect interesting relationships between price and profitability could result in the discovery of a relationship that would be used to train a supervised learning system that recommends items to a customer. For the retailer, combining these two systems could form a virtuous cycle of retail knowledge, each improving the performance of the other over time.

Hypotheses Generation and Scoring

A *hypothesis* in science is a testable assertion based on evidence that explains some observed phenomenon or relationship between elements within a domain. The key concept here is that a hypothesis has some supporting evidence or knowledge that makes it a plausible explanation for a causal relationship. It isn't a guess. When a scientist formulates a hypothesis as an answer to a question, it is done in a way that allows it to be tested. The hypothesis actually has to predict an experimental outcome. An experiment or series of experiments that supports the hypothesis increases confidence in the ability of the hypothesis to explain the phenomenon. This is conceptually similar to a hypothesis in logic, generally stated as "if P then Q", where "P" is the hypothesis and "Q" is the conclusion.

In the natural sciences we conduct experiments to test hypotheses. Using formal logic we can develop proofs to show that a conclusion follows from a hypothesis (or that it does not follow). In a cognitive computing system we look for evidence—experiences and data or relationships between data elements—to support or refute hypotheses. That is the basis for scoring or assigning a confidence level for a hypothesis. If a cognitive computing hypothesis can be expressed as a logical inference, it may be tested using mechanical theorem proving algorithms. Typically, however, cognitive computing applications solve problems in domains with supporting data that are not so neatly structured. These domains, like medicine and finance, have rich bodies of supporting data that are better suited to statistical methods like those used in scientific experimental design.

Figure 2-2 shows a virtuous cycle of hypotheses generation and scoring. Here the plural "hypotheses" indicates that cognitive systems can generate multiple hypotheses based on the state of data in the corpus at a given time. In general, these can then be evaluated and scored in parallel. In a system such as IBM's Watson, for example, 100 independent hypotheses may be generated in a single cycle. Each may be assigned to a separate thread or core for scoring. That

enables the system to leverage parallel hardware architectures and optimize parallel workloads. Of course, conceptually a system could generate and score all hypotheses sequentially. The parallel architecture and workload design of Watson is discussed in detail in Chapter 9, "IBM's Watson as a Cognitive System."

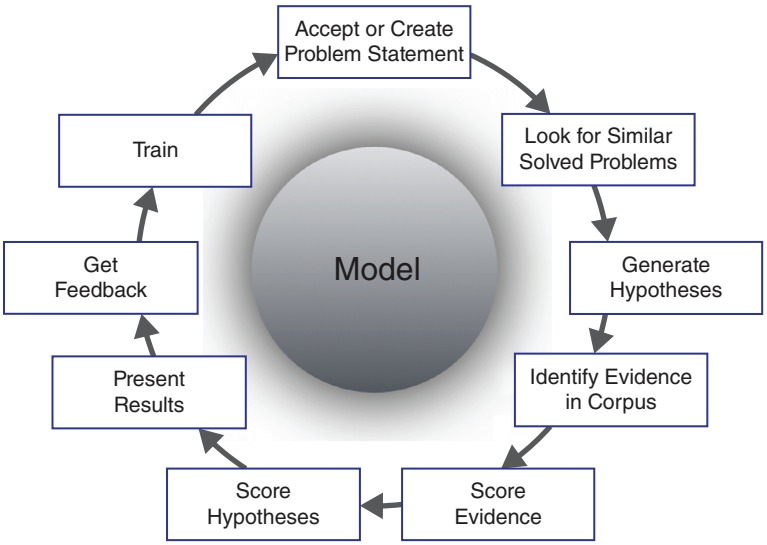

Figure 2-2: The continuous machine learning process

Hypothesis Generation

The discussion about the scientific method said that a hypothesis is formulated to answer a question about a phenomenon based on some evidence that made it plausible. The experimental process is designed to test whether the hypothesis applies in the general case, not just with the evidence that was used to develop the hypothesis. In a typical cognitive computing system, there are two key ways a hypothesis may be generated. The first is in response to an explicit question from the user, such as "What might cause my fever of 102 and sore throat?"

In this scenario, the cognitive computing application must look for plausible explanations. It could, for example, start by presenting all the possible conditions in which you might expect to see these symptoms. (Each condition would be a candidate hypothesis explaining the symptoms.) Alternatively, it may recognize that there are too many answers to be useful and request more information from the user to refine the set of likely causes. This approach to hypothesis generation is frequently used when the goal is to detect a relationship between cause and effect in a domain in which there is a known set of causes and a known set of effects, but there are so many combinations that the mapping of all causes to all effects is an intractable problem for humans to solve. Typically, this type of cognitive

computing system will be trained with an extensive set of question/answer pairs. The process of hypothesis generation is one of generating candidate hypotheses that appear to have a similar relationship to the user's question as the relationship between known correct question-answer pairs from the training data set.

The second type of hypothesis generation does not depend on a user asking a specific question. Instead, the system constantly looks for anomalous data patterns that may indicate threats or opportunities. Detecting a new pattern creates a hypothesis based on the nature of the data. For example, if the system is monitoring network sensors to detect threats, a new pattern may create a hypothesis that this pattern is a threat, and the system must either find evidence to support or refute that hypothesis. If the system is monitoring real-time stock transactions, a new pattern of buying behavior may indicate an opportunity. In these systems, the type of hypotheses that will be generated depends on assumptions of the system designers rather than on the actions of the users. Both types of application have the system generate one or more hypotheses based on an event, but in the first case, the event is a user question, and in the second it is driven by a change in the data itself.

Hypothesis Scoring

At this point, you have seen how cognitive computing systems build a corpus of relevant data for a problem domain. Then, in response to a user question or change in the data, the system generates one or more hypotheses to answer a user's question or explain a new data pattern.

The next step is to evaluate or score these hypotheses based on the evidence in the corpus, and then update the corpus and report the findings to the user or another external system. Hypothesis scoring is a process in which the representation of the hypothesis is compared with data in the corpus to see what evidence exists to support the hypothesis, and what may actually refute it (or rule it out as a valid possible explanation). In fact, scoring or evaluating a hypothesis is a process of applying statistical methods to the hypothesis-evidence pairs to assign a confidence level to the hypothesis. The actual weights that are assigned to each piece of supporting evidence can be adjusted based on experience with the system and feedback during training and during the operational phase. If none of the hypotheses score above a predetermined threshold, the system may ask for more evidence (a new diagnostic blood test, for example) if that information could change the confidence in the hypothesis. Techniques for measuring the proximity of two data elements or structures for pattern matching, such as the fit between two hypothesis-evidence pairs, generally rely on a binary vector representation (such as a sparse distributed representation [SDR]) that can be manipulated using matrix algebra with readily available tools.

The generation/scoring loop may be set to continue until a user is satisfied with the answer or until the system has evaluated all options. The next section

shows how machine learning algorithms guide this loop in practice. You need to remember that although the system generates and scores the hypotheses based on the evidence, in most cases the user actually provides feedback on the answer that may be viewed as scoring the process. This ongoing interaction between user and system provides a virtuous cycle of learning for man and machine.

Presentation and Visualization Services

As a cognitive computing system cycles through the hypothesis generation and scoring cycle, it may produce new answers or candidate answers for a user. In some situations, the user may need to provide additional information. How the system presents these findings or questions will have a big impact on the usability of the system in two ways. First, when presenting data supporting a hypothesis such as a medical diagnosis or recommended vacation plan, the system should present the finding in a way that conveys the most meaning with the least effort on the part of the user and support the finding with relevant evidence. Second, when the system requires additional details to improve its confidence in one or more hypotheses, the user must present that data in a concise and unambiguous way. The general advantage for visualization tools is their capability to graphically depict relationships between data elements in ways that focus attention on trends and abstraction rather than forcing the user to find these patterns in the raw data.

Following are three main types of services available to accomplish these goals.

- Narrative solutions, which use natural language generation techniques to tell a story about the data or summarize findings in natural language. This is appropriate for reporting findings or explanations about the evidence used to arrive at a conclusion or question.
- Visualization services present data in nontext forms, including:
  - Graphics, ranging from simple charts and graphs to multidimensional representations of relationships between data.
  - Images, selected from the data to be presented or generated from an underlying representation. (For example, if feature extraction detects a "face" object, a visualization service could generate a "face" or pictograph from a standard features library.)
  - Gestures or animation of data designed to convey meaning or emotion.
- Reporting services refers to functions that produce structured output, such as database records, that may be suitable for humans or machines.

Some data may naturally lend itself to only one of these options, but often the system may convey the same information in multiple formats. The choice of

which tools or formats should ultimately be made by the user, but as a system learns preferences based on user choices in ongoing interactions, it can use the most commonly requested formats as defaults for individual users.

Infrastructure

The infrastructure/deployment modalities layer referred to in Figure 2-1 represents the hardware, networking, and storage foundation for the cognitive computing application. As noted in the discussion on emerging neuromorphic hardware architectures in Chapter 14, "Future Applications for Cognitive Computing," most cognitive computing systems built in the next decade will primarily use conventional hardware. The two major design considerations for cognitive computing infrastructure decisions are:

- **Distributed Data Management**—For all but the smallest applications, cognitive computing systems can benefit from tools to leverage distributed external data resources and to distribute their operational workloads. Managing the ongoing ingestion of data from a variety of external sources requires a robust infrastructure that can efficiently import large quantities of data. Based on the domain, this may be a combination of structured and unstructured data available for batch or streaming ingestion. Today, a cloud-first approach to data management is recommended to provide maximum flexibility and scalability.
- **Parallelism**—The fundamental cognitive computing cycle of hypothesis generation and scoring can benefit enormously from a software architecture that supports parallel generation/scoring of multiple hypotheses, but performance ultimately depends on the right hardware. Allocating each independent hypothesis to a separate hardware thread or core is a requirement in most cases for acceptable performance as the corpus scales up and the number of hypotheses increases. Although performance improvements should be seen within the system as it learns, the rate of data expansion in the corpus generally outpaces this performance improvement. That argues strongly for the selection of hardware architecture that supports relatively seamless expansion with additional processors.

Summary

Designing a cognitive system requires bringing together many different elements, beginning with the corpus of data. To create a system that provides an interface between humans and machines means that the system must learn from data and human interaction through an iterative process. The system must be designed so that users can interact with the system both through a language or a visual interface.