Chapter 9

Contextual Reasoning

In the last chapter we introduced the concepts of context and contextual information. In this chapter, we look more deeply at how smartphones can obtain and process this type of information. There are a number of terms that have been suggested to encompass the idea of processing contextual information, such as contextual thinking [1] and contextual intelligence [2]. We have chosen *contextual reasoning* [3], but all of these terms are more or less equivalent. We will present several examples of the techniques used for contextual reasoning, though we will not attempt to be exhaustive in our coverage. To aid those wishing to go deeper in this topic, we will point out ample references where additional information can be found.

9.1 WHAT IS CONTEXTUAL REASONING?

We begin with a formal definition. *Contextual reasoning is the process of forming higher level inferences about context from lower level information*. "Higher" and "lower" are relative terms in this definition, where the higher levels are more general and abstract, and the lower levels are more specific details that individually don't provide the "big picture" of a context. In colloquial terms, it is the process of seeing the forest through the individual trees.

Figure 9.1 illustrates this process conceptually. Known as the "context pyramid," this figure divides the contextual reasoning process into six different abstraction levels, although in practice there may be fewer or more levels, depending on the application. The first level represents the raw data received from sensors, whether we speak of hardware sensors like accelerometers and gyroscopes, or "soft sensors" like a GNSS receiver or application code that detects various states of the smartphone (e.g., low battery warning). In the next level of abstraction, physical parameters are derived from the raw sensor data, such as the speed that the user is moving or decibel level recorded by a microphone. In the third level (which may be optional in some applications), various features or

patterns may be extracted from the lower levels, such as combining successive parameters to calculate a moving average or variance of the parameters.

The next three levels represent the "heart" of contextual reasoning. We move from the realm of numeric data to more semantic representations of context. In the first such level, simple contextual descriptors are inferred from the lower-level data, such as the motion pattern of the user (e.g., standing and walking), or a semantic representation of the user's position (e.g., at work). In the next level, several simple contextual descriptors may be combined to infer a higher-level context, such as the user's current activity. Finally, all of the available contextual information, including information from external sources, may be combined to a final level, which (if successful) can be described as a *rich context.*[24] Ideally this should be expressed in natural language in a form that approaches prose. This is the final result of asking and answering the questions from the journalistic framework described in Chapter 8.

9.2 A HYPOTHETICAL EXAMPLE

To make these concepts more concrete and to provide a geospatial computing example, consider the following low-level information presented in Table 9.1 from a hypothetical example.

A logical high-level inference that might be made from this information is, "Mary is walking to work." This could probably be considered an activity-level descriptor (level 5) in terms of the context pyramid as shown in Figure 9.1. If additional contextual details were to be added, such as the time of day, weather conditions, or any interesting events or details that occurred in the recent past, then it might rise to the level of rich context. As one can imagine, the exact boundaries between these levels are blurry, and the context pyramid should be considered only as a tool for understanding the process of contextual reasoning.

Note that probably several intermediate steps were required in the process of inferring, "Mary is walking to work." For example, we might have calculated that at Mary's current position, she is 500m away from the FGI and that her heading indicates she is going toward the street that leads to FGI.

Clearly humans make inferences like "Mary is walking to work" all the time. It is one of the main functions of our brain to perform this kind of reasoning. Sometimes there are a few intermediate steps performed in the subconscious, and the inference is performed instantaneously. Other times, there may be many intermediate steps, and the inference might require careful analysis of all the available information. It might even require hypothesizing and probing for additional information to confirm a hypothesis.

---

[24] In using the adjective "rich," we invoke the concept from AI of a *rich object*, which is defined as "an object which cannot be completely described or represented but about which assertions can be made" [33].

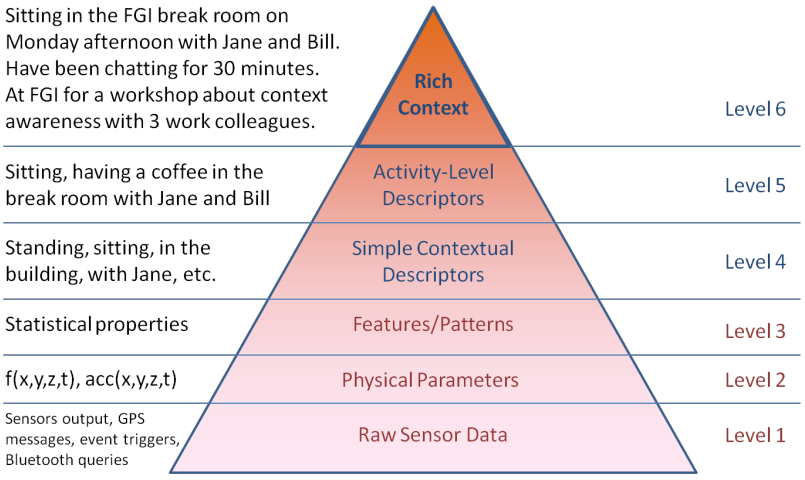


Figure 9.1 The context pyramid.

Table 9.1

Example of Contextual Information, Its Source, and Corresponding Level

Information	*Source*	*Level*
Mary is located at 60.161 N, 24.542 E.	GPS	Level 2
Mary has a heading of 170 degrees.	GPS	Level 2
Mary is walking.	Motion classifier	Level 4
Mary works at FGI.	Personal profile	External
FGI is located at 60.161 N, 24.546 E.	Significant location DB	External

The same is true for computer-based contextual reasoning. Sometimes there is enough available information and the inference is simple enough to be performed in realtime without any human interaction. In other cases, it can be extremely complex (probabilistic in nature), and require significant processing power. It may even require additional input from the user (i.e., human-computer interaction).

Consider the following additional information related to the above example:

It is 6 p.m. on a Sunday evening.
Mary normally works from 9 a.m.–5 p.m. on weekdays.

From this additional information, a computer-based contextual reasoning program might output that there is a 60% chance Mary is walking to work and a 40% chance she is walking somewhere else. Due to this ambiguity, the program queries the system for additional information:

Mary is walking with her dog.
Mary lives at 60.10 N, 25.00 E.

After this additional information and a few intermediate calculations, the program outputs:

Mary is walking her dog, near her workplace and headed away from her home.

We can see from this example that there are different inferences that can be made from different subsets of low-level information. Furthermore, we see that, in general, the richer the set of information we have to formulate our inferences, the better chance we have of generating a correct one.

Finally, observe that contextual reasoning can also be thought of as a process of *encapsulation* of information. For many applications, it is not interesting to the user nor directly useful for the application that "FGI is located at 60.10 N, 25.10 E" or (worse) that "the standard deviation in Mary's speed for the past 5 seconds is 0.58 m/s." What the application (or user) needs to know is "what is Mary doing," "who is she with," "in what manner is this activity playing out," and similar higher level questions. Thus, contextual reasoning can be alternatively thought of as the process of transforming the multitude of available information into a more concise and useful contextual result, where the definition of "useful" is highly dependent on the desired application.

9.3 WHAT ARE THE METHODS OF CONTEXTUAL REASONING?

The primary tool for making contextual inference is *machine learning*, also known as *statistical learning* or *pattern recognition.*[25] Machine learning is deeply rooted in probability theory, decision theory, and information theory. Unfortunately, we can't cover all of these topics in this chapter, but we refer those readers who are not familiar with these topics to introductory texts on machine learning, where these topics are adequately covered. In truth, it could take years of careful study to deeply understand the state-of-the-art in machine learning and contextual reasoning. Fortunately, however, there are several simple, basic techniques that work well for many applications. This section is intended to give a gentle

[25] Historically, pattern recognition grew out of engineering disciplines, whereas machine learning is the term adopted by most computer scientists [5]. For obvious reasons, statistical learning is the term favored by statisticians. O'Connor humorously noted that at Stanford University there are two almost identical courses offered: "machine learning" by the computer science department and "statistical learning" taught by the statistics department [6].

introduction to some of the common techniques of machine learning that are applied to contextual reasoning.

9.3.1 Introduction to Machine Learning

According to Mitchell et al., "machine learning research seeks to develop computer systems that automatically improve their performance through experience" [4]. In the domain of contextual reasoning, this means that, given the tools of machine learning, the more the system is used, the better it should be able to reason about the context the user is in. The word *learning* is appropriate because the system adaptively learns how to recognize different contexts that it encounters, in the same way that a small child (or even adult) learns to recognize different situations based on his or her experiences. In both cases (machine and human), there is a set of inputs or stimuli that are used to reason that a certain situation or context exists. The process of learning (which happens naturally and almost automatically for humans) is figuring out the optimal way to transform a set of inputs into the correct output—in our case, the context.

In general, the output of a machine learning algorithm can be of two types: 1) one or more continuous variables represented by real numbers, 2) a discrete class, which is normally represented by a textual descriptor (e.g., green, male, good, and happy) or an integer or binary code, from a finite set of classes. When the output is of the continuous type, the machine learning task is known as *regression*. When it is of the discrete type, it is known as *classification*. Since context is primarily of a discrete nature, we will focus mostly on classification in this chapter.

In classification, we can understand the learning problem as a process of learning a function, $f(\cdot)$, that maps a vector of inputs $\boldsymbol{x} = (x_1, x_2,...x_n)$ to the correct output $y \in Y=\{y_1, y_2,...y_m\}$. This is expressed symbolically as:

$$f : \boldsymbol{x} \rightarrow y \tag{9.1}$$

Note that the function need not be a deterministic one. It may consist of probability distributions of the input and output variables, describing a stochastic relationship between them. Ultimately, however, a classifier chooses a single class based on the function's inputs and using a particular selection criterion, such as *maximum likelihood*.

Machine learning techniques mostly fall into one of two categories based on how the function f is learned: 1) *supervised learning* and 2) *unsupervised learning*. In supervised learning, a set of "training data" is used. The training data is a limited set of input data for which the correct or optimal output is known [i.e., $\mathbf{s} = (\boldsymbol{x}, y)$]. This is sometimes known as labeled data because in most cases a human has manually labeled the dataset with the correct output. Therefore, it is usually laborious and costly to obtain such data. On the other hand, in unsupervised learning no training data are used, and the goal is to recognize

implicit patterns in the data. There is, in fact, a third category of machine learning, known as *semisupervised learning*, in which both labeled and unlabeled data are used. The aim in semisupervised learning is to combine techniques from supervised and unsupervised learning and use all available data (labeled and unlabeled) to achieve a better learning result. In this book, we will mainly focus on supervised learning techniques, as they are the most commonly used in context recognition.

Most of the techniques in machine learning have been developed to deal with data samples that are independent from one another. For example, in a machine learning algorithm to classify credit card applications as "high-risk" or "low-risk," it is reasonable to assume that one credit card application does not influence the risk level of another credit card application. In contextual reasoning, however, our data samples mostly (if not exclusively) come from *time-series data*, where the assumption of independence does not strictly hold.

We can understand this intuitively from the fact that what a person is doing at one moment usually affects what he or she does at the next moment. In other words, a data sample at any given time epoch is dependent on the values of the data at prior time epochs, especially those epochs immediately prior to the given epoch. To a certain extent, we can choose to ignore these dependencies, in order to use a machine learning model with a more simple structure, but the best performance will be achieved when these time dependencies are taken into account. The subset of techniques from statistics and machine learning that work with this kind of data is known as *time-series analysis* [7] or *sequential machine learning* [8, 9].

There are certainly many such techniques to choose from, each having relative advantages and disadvantages. Table 9.2 presents a condensed list of techniques from machine learning that could be applied to contextual reasoning. It is not exhaustive, but provides a broad overview of the most important machine learning techniques for contextual reasoning. A subset of these techniques will be discussed at an introductory level in the sections below.

It is tempting when presented with a long list of machine learning techniques to want to find and focus on the "best" one, applying it indiscriminately in any given domain. The *no free lunch theorem for supervised learning* states, however, that no single machine learning algorithm performs better than any other across all problems. Thus, it is unavoidable that some process of comparison or analysis must be carried out for any particular domain before the most appropriate machine learning technique is chosen.

9.3.2 Naïve Bayes' Classifiers

Before we turn to examples of sequential machine learning techniques, we start with an example of a simple machine learning technique, in order to motivate more complex concepts later on. This simple yet surprisingly effective classifier is called the *naïve Bayes' classifier*. It falls within a larger set of powerful graphical

probability models, called *Bayesian networks*, but takes advantage of a few simplifying assumptions.

Table 9.2

List of Machine Learning Techniques and Associated References

Supervised Learning	*References*
Used with sliding window method:	
Naïve Bayes'	[10-11] [13]
Bayesian networks	[24]
Decision trees	[25] [42]
LDA/QDA	[34-36] [38-40]
Logistic regression	[40-42]
SVMs/kernel machines	[26-28] [34] [41]
Neural networks	[40] [42]
Used directly on time-series data:	
HMMs	[18-19] [23] [35] [43]
Conditional random fields	[29-32] [34] [43]
Unsupervised Learning	*References*
Clustering	[37]
Principal component analysis	[36]
Self-organizing maps	[44-46]

First, we assume that all data samples are *independent and identically distributed* (*iid*). In other words, we ignore the time dependence between the data samples, and we assume that the set of processes producing the data are stable throughout the dataset (i.e., stationary processes). Second, although the data samples may be multivariate, we assume that all of the measured variables are conditionally independent from one another given the class. In many (if not most) applications, these assumptions are not strictly true, but the time dependencies and the possible dependencies between the measured variables are simply ignored in order to reduce the problem to a more tractable solution, hence the name "naïve."

Despite these simplifications, naïve Bayes' classifiers have been shown to perform well in a number of machine learning domains, including document classification [10], image classification [11], and activity recognition [12, 13]. We will use them as a starting point in our discussion of contextual reasoning because they illustrate a number of fundamental concepts before we move on to more complex machine learning techniques.

The basic structure of a naïve Bayes' classifier is shown graphically in Figure 9.2, where the unshaded node represents a class set $C_k = \{c_1,c_2,\ldots,c_n\}$ and the shaded nodes represent a d-dimensional input vector $\boldsymbol{x} = (x_1,x_2,\ldots,x_d)$. The arrows represent the dependencies between the classes and the input vector. The lack of arrows between the variables in the input vector indicates that they are conditionally independent given the class.

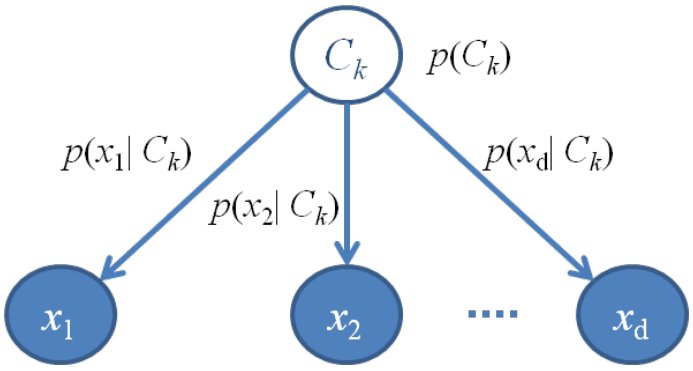


Figure 9.2 Structure of the naïve Bayes' classifier.

We assume that, in general, the class cannot be observed directly, whereas the input vector can be; therefore, we would like to classify a dataset into classes, according to this input vector. In other words, we want to know the posterior probability distribution of C_k given $\boldsymbol{x}$, which according to Bayes' theorem is:

$$p(C_k \mid \boldsymbol{x}) = \frac{p(C_k)p(\boldsymbol{x} \mid C_k)}{p(\boldsymbol{x})} = \frac{p(C_k)p(\boldsymbol{x} \mid C_k)}{\sum_k p(\boldsymbol{x} \mid C_k)p(C_k)} \tag{9.2}$$

In order to evaluate the likelihood $p(\boldsymbol{x}|C_k)$, we would ordinarily need to compute the product:

$$p(x_1 \mid C_k)\prod_{i=2}^{d} p(x_i \mid x_1,...,x_{i-1}, C_k),$$

expressing the conditional dependence among the input variables within each class, which may be very difficult if the number of input dimensions is large. By assuming that the input variables are conditionally independent given the class, the product reduces to:

$$p(\mathbf{x}|C_k) = \prod_{i=1}^{d} p(x_i|C_k) \tag{9.3}$$

The most common way to evaluate $p(\boldsymbol{x}|C_k)$ is to use a training dataset, where the values are of $\boldsymbol{x}$ are labeled with the correct class (i.e., supervised learning). If $\boldsymbol{x}$ is composed of discrete variables (e.g., with multinomial distribution), $p(x_i|C_k)$ can be estimated as the frequency that x_i takes on a particular value for a particular class; stated another way:

$$p\left(x_i = x_j \middle| C_k = C_m\right) = \frac{\text{count}(x_i = x_j, C_k = C_m)}{\text{count}(C_k = C_m)} \tag{9.4}$$

If $\boldsymbol{x}$ is composed of continuous variables, then other methods must be used to estimate this class conditional density, such as a histogram, kernel estimator, or a k-nearest-neighbor approach. Alternatively, one can use parametric methods, especially if certain assumptions about the distribution can be made (e.g., that it is Gaussian). For more details on parametric and nonparametric estimation methods, see [14].

The prior probabilities of the classes, $p(C_k)$, can also be estimated using frequency counts:

$$p(C_k = C_m) = \frac{\text{count}(C_k = C_m)}{N} \tag{9.5}$$

where N is the total number of training samples. Thus, $p(C_k = C_m)$ is simply the fraction of the training samples that are labeled with class C_m. In some cases, however, it may be that the training data disproportionately represent certain classes compared to the real-world case. Therefore, one can use domain knowledge to adjust the class priors appropriately. For example, in some applications, it may be appropriate to use equal probabilities for each class. Just be sure that the probabilities over all classes sum to one.

Note that in (9.2), the denominator will be the same for all classes. Since our ultimate goal is to choose the most likely class [i.e., the largest posterior probability, $p(C_k|\boldsymbol{x})$], it is not necessary to actually compute the value of the denominator. Given data sample $\boldsymbol{x_j}$, one can find the largest value of $p(C_k)p\left(\boldsymbol{x_j}\middle|C_k\right)$ and classify the sample into the respective class given by the arguments.

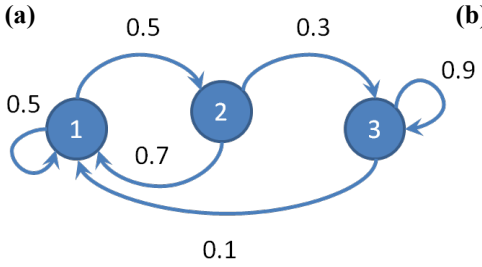
9.3.3 Hidden Markov Model (HMM)–Based Classifiers

In the next section, we turn to a slightly more complex but more powerful machine learning technique that uses HMMs. Generally speaking, HMM-based classifiers will perform better than simple naïve Bayes' classifiers for time-series data because they are able to model the time dependency in the data. For this reason, HMMs are one of the most popular machine learning techniques for contextual reasoning.

9.3.3.1 Markov Chains and the Markov Property

First, we introduce the concept of a *Markov chain*, a mathematical system used to model stochastic processes that exhibit a *Markov property*, which will be

described shortly. These processes are usually interpreted as belonging to a system that can be, at any given time, in one state from a set of possible states, S. In this chapter, we will consider only discrete state processes with a finite set of possible states, S = {S_1, S_2,…S_M}. The system transitions from one state to another in a stochastic manner, meaning that the transitions can be analyzed using probabilities. Such a system is represented graphically in Figure 9.3(a), where the probabilities governing the evolution of states {1, 2, 3} are shown as arrows between the nodes in the graph or as loops back onto themselves.



(b)

S_i \ S_j	State 1	State 2	State 3
State 1	0.5	0.5	0.0
State 2	0.7	0.0	0.3
State 3	0.1	0.0	0.9

Figure 9.3 A Markov chain and associated transition probability matrix.

For example, if the system is in state 1 at time *t*, it has a 50% probability of transitioning at time *t*+1 to state 2 and a 50% probability of remaining in state 1. Note that the lack of an arrow or loop indicates a zero probability of that particular transition. The full set of transition probabilities can also be expressed as an *M* x *M* matrix, where *M* is the number of states, as shown in Figure 9.3(b) for this example Markov chain. These correspond to conditional probabilities, where element *ij* is the conditional probability of transitioning from the *i*th state to the *j*th state at any epoch *t*.

When a Markov chain can be fully specified in this manner, it is said to be a first-order Markov process. The two-dimensional structure of the transition probabilities ensures that all of the information available to predict the next state of the process is contained in the present state. In probability terms, this means that the conditional probability of the system being in state s_j at time *t*+1, given knowledge that the system is in state s_i at time *t*, is equal to the conditional probability of the same outcome at $t + 1$, given the same knowledge of the system's state at time t plus information of its state at time $t - 1$ and earlier; i.e. stated another way:

$$P\big(S(t+1) = s_j \big| S(t) = s_i\big) = P\big(S(t+1) = s_j \big| S(t) = s_i, S(t-1) = s_k, \dots, S(1) = s_l\big) \quad (9.6)$$

When (9.6) holds for a system, we say it has the *first-order Markov property*. The second-order Markov property is when the state at $t + 1$ depends on the current

state and the state at $t - 1$. This, however, would require a three-dimensional transition probability matrix ($M \times M \times M$) to be fully specified. In general, it is also possible for a system to have an nth-order Markov property, which would require an n-dimensional transition probability matrix. Only first-order Markov processes will be considered further in this chapter.

In other words, a (first-order) Markov process' exact "path" to a particular state does not provide any more useful information than the current state of the system. An example of this kind of process would be a chess game, where in order to assess the possible next move, the current configuration of the board is just as useful as the exact sequence of moves in the game up to the current configuration.

9.3.3.2 Hidden States and Observable Signals

There are two differences between an ordinary Markov model and a HMM. The first is that in HMMs, the state of the system at any given moment is not directly observable (i.e., is hidden). Second, for each transition of states, the system "emits" an observable signal, which is stochastically correlated with the unobservable state. All that one can "know" about the system's state is contained in the observed signal (which may have several components), and one can make probabilistic inferences about the state based on the correlation between the states and the observations.

As an example, consider again a chess game. Suppose now that an observer is far away and cannot make out the board but can see the players' faces and gestures and also knows something about their characters. For example, based on previous experience, this observer knows that player 1 usually smiles broadly after making a good move and often groans when she realizes she made a bad move. Player 2 is much more calm and relaxed during the game but almost always smiles the moment he realizes he has checkmate.

This could possibly be modeled using a HMM where the states are {*game even, player 1 winning, player 2 winning, player 1 wins, player 2 wins*}. The observed signals are the smiles, gestures, and sounds of the players, but the actual state is "hidden" because the board is far away. Another variant of this example would be where the observer just doesn't know the rules of chess, so is unable to assess the state, except at the very beginning and very end.[26]

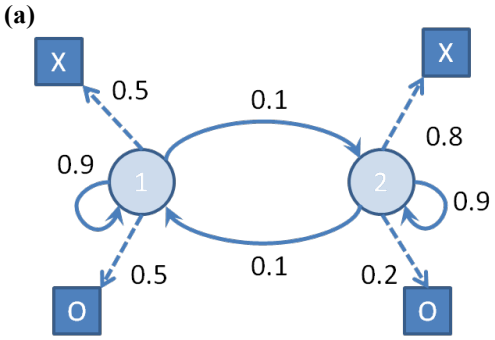
HMMs can either be discrete or continuous, depending on the nature of the observed signal. For simplicity we will consider only discrete HMMs in this chapter, but many approaches have been developed to handle continuous observation signals, for example, by describing the signal parametrically. See [19] for examples.

Figure 9.4(a) shows a graphical representation of a HMM, similar to that of a Markov chain but adding the observation signal. In this example, there are only

---

[26] This would be an example of a partially HMM. See [15] for details.

two possible states {1, 2}, and the lighter shading indicates they are not directly observable. In this particular example, the transition probabilities constrain the system to only occasionally change states, since most of the probability for each state (90%) is concentrated in the loops. These transition probabilities are shown as before in Figure 9.4(b). In addition, there are two possible output signals Y = {X, O}. In state 1, these are expressed with equal probability, whereas state 2 has a tendency to express more X's than O's (80% vs. 20%). These "emission" probabilities are shown in Figure 9.4(c). In general, for discrete HMMs the emission probabilities are given by an MxN matrix, where M is the number of states and N is the number of observation symbols.[27]



(b)

S_i \ S_j	State 1	State 2
State 1	0.9	0.1
State 2	0.1	0.9

(c)

S_i \ Y_i	X	O
State 1	0.5	0.8
State 2	0.5	0.2

Figure 9.4 A Markov chain and associated transition probability matrix.

9.3.3.3 Machine Learning Tasks Using HMMs

Now that we have laid out the basic concepts of HMM, let us explore how they are used in machine learning and contextual reasoning. From here forward it will be important to use clearly defined mathematical notation for HMMs in order to keep the text concise. We adopt the notations listed in Table 9.3, which have already been partially used above.

There are primarily four machine learning tasks that are commonly performed with HMMs, and each task has its own standard algorithm. These tasks are described as follows:

[27] For continuous HMMs, the emission probabilities might be expressed by a parametric model (e.g., a Gaussian with different parameters for each state).

- **Supervised model learning.** Given a sequence of observations Y_1^T that are labeled with the true sequence of states S_1^T, estimate the model parameters λ.[28]
- **Unsupervised model learning.** Given a sequence of observations Y_1^T and an initial guess of the model parameters λ_0, find the set of parameters λ_n that optimally matches the observations.
- **Online recognition.** Given a model λ and a sequence of observations Y_1^t, find the most likely state s_i at time t. Repeat for each epoch, t=1,…,T to obtain the sequence of states S_1^T. This is also known as sequential processing.
- **Offline recognition.** Given a model λ and a sequence of observations Y_1^T, find the most likely sequence of states S_1^T [i.e., maximum $\Pr(S_1^T|Y_1^T,\lambda)$]. This is also known as batch processing.

Table 9.3

Mathematical Notations of the HMMs

Notation	*Description*
T	The number of elements in a sequence of observations.
Y_1^T	A sequence of observations, indexed from 1 to T.
M	The number of possible states in the model.
$S = \{s_1, s_2, \dots, s_M\}$	The set of possible states.
$S(t)$	The state at time t.
S_1^T	A sequence of states, indexed from 1 to T.
N	The number of possible observation symbols.
$Y = \{y_1, y_2, \dots, y_N\}$	The set of possible observation symbols.
$Y(t)$	The observation symbol at time t.
$a_{ij} = \Pr(S(t+1) = s_j \mid S(t) = s_i)$	The conditional probability that the system will be in state s_j at time t+1, given the state s_i at time t. Also called the transition probability from state i to j.
$b_k = \Pr(Y(t) = y_k \mid S(t) = s_j)$	The probability that state s_j will emit observation symbol y_k. Also called the emission probability for y_k and s_j.
$B = \{b_j(k)\}$ for $j = 1, \dots, M$ and $k = 1, \dots, N$	The set of emission probabilities for all observation and symbols and states, also called the emission probability matrix.
$\pi = Pr\{S(1) = s_j\}$ for $j = 1, \dots, M$	The set of initial state probabilities for each state s_j.
$\lambda = \{A, B, \pi\}$	The complete parameter set of the model.

There are two subtle but important differences between task 3 and task 4. First, task 4 is an offline task, so the algorithms used to achieve it can use the full set of observations Y_1^T at any time in its operation, whereas task 3 can only use

[28] Note that we have just broken the assumption about the state of the system not being directly observable. This is true only for the model learning phase (i.e., training). During the recognition phase, we still assume the states are hidden.

past observations. Second, task 3 operates at each epoch individually and will produce an optimal choice of state at that particular epoch (given the available observations), whereas task 4 chooses the optimal sequence S_1^T, which has the highest probability of producing the observations Y_1^T. In other words, they have two distinct choices of optimality criterion, which can sometimes lead to differing results.

Supervised Model Learning

This is the most straightforward of the four tasks. First, to estimate the transition probability matrix A, we simply need to count up the occurrences of each transition type i→j (including where j=i) for all i,j={1,…,M}. The transition probabilities are then estimated as:

$$a_{ij} = \frac{\text{count}(S(t) = s_i, S(t+1) = s_j)}{\text{count}(S(t) = s_i)} \tag{9.7}$$

This will give us M^2 transition probabilities, allowing us to fully specify the matrix A. In practice, however, some of these transition probabilities will be zero. This *could* be because the probability of that particular transition is actually zero, or it could be simply because the transition is not represented in the limited training sequence S_1^T. Therefore, it is good practice to manually inspect the matrix A and use one's domain knowledge to determine if some possible transitions are not represented in the dataset when they are indeed possible. One approach to rectify this problem is to add a *pseudocount* to each such transition. The value of the pseudocount can either be one (Laplace's rule) or can be set to some other positive integer value based on domain knowledge.

Next, to estimate the emission probability matrix B, we count for each state the number of times each observation symbol has been expressed. The emission probability is then estimated as:

$$b_j(k) = \frac{\text{count}(Y(t) = y_k, S(t) = s_j)}{\text{count}(S(t) = s_j)} \tag{9.8}$$

Again we can use Laplace's rule or some other pseudocount value, based on domain knowledge, in order to correct any probabilities that would otherwise be unreasonably set to zero.

Last, we must estimate the initial state probabilities that should be specified in π. In practice, the methods for setting this parameter are application-dependent. The simplest approach is to set it to an equal value for each state that sums to one over all states (i.e., $1/M$). If for some particular application, it is more likely that the HMM is initialized in some particular state or states, the values for these states can be adjusted according to domain knowledge (maintaining that the probabilities

over all the states sum to one). Another possibility is when the initial state of an HMM is known with great certainty, the value for that state can be set to one and all other initial state probabilities set to zero. Note that no matter how parameter π is specified, its importance diminishes as t increases, so even the simple $1/M$ approach may be adequate.

Unsupervised Model Learning

In this task, we would like to estimate the parameters of the HMM based on an unlabeled observation sequence Y_1^T. In order to do so, we need an initial guess of the parameters, λ_0. The standard algorithm for optimizing the parameters based on this initial guess and an observation sequence is called the *Baum-Welch algorithm* or the *forward-backward algorithm*. A detailed treatment of this algorithm (which includes several variants) is beyond the scope of this chapter, but we only mention here that it is an expectation–maximization (EM) procedure, where an improved estimate of the model parameters is given after each iteration, until converging to a local maximum of $Pr(Y_1^T|\lambda)$. The result varies based on the initial guess of the parameters, so one approach is to run the algorithm a number of times with randomly generated initial guesses (subject to reasonable constraints). This won't necessarily yield a globally optimized set of parameters, but no finite time approach to obtain a global maximum of $Pr(Y_1^T|\lambda)$ is known. More details about the Baum-Welch algorithm can be found in [14] or [19].

Online Recognition

Given a HMM defined by parameters λ and a history of observations Y_1^T, the goal of this task is to determine the most likely state s_i at time t. In other words, we need to find the state that gives the maximum probability $Pr(S(t)|Y(t),\lambda)$ from the set of all possible states, also known as the maximum a posteriori (MAP) estimate of S(t):

$$\hat{S}(t)_{MAP} \equiv \arg\max_i \Pr\left(S(t) = s_i | Y(t), \lambda\right) \tag{9.9}$$

In order to evaluate this probability, we use an identity called the Chapman–Kolmogorov equation and the Markov property to obtain a prediction equation:

$$p(S(t) = s_j \mid Y(t-1)) = \sum_{j=1}^{M} \sum_{i=1}^{M} a_{ij}\, p(S(t-1) = s_i \mid Y(t-1)) \delta(s_j - s_i) \tag{9.10}$$

where $\delta(s_j - s_i)$ is the Dirac delta measure that is equal to zero for $s_i \neq s_j$ and one for $s_i = s_j$ [16]. Then, given a new observation $Y(t)$, we use Bayes' rule to obtain the updated equation:

$$p(S(t) \mid Y(t) = k) = \frac{b_j(k)\, p(S(t) = s_j \mid Y(t-1))}{\sum_{i=1}^{N} b_i(k)\, p(S(t) = s_i \mid Y(t-1))} \tag{9.11}$$

These two equations form the basis of the prediction and update steps of this recognition algorithm, respectively. At each time epoch, the prediction density $p\big(S(t) = s_j \big| Y(t-1)\big)$ is calculated and then updated using the new observation and (9.11). This technique is often known as Bayesian optimal filtering or recursive Bayesian estimation. In some references, it has been called the grid-based method, presumably because the transition probabilities are represented as a grid (i.e., matrix) [16].

Offline Recognition

The goal of this task is to find the most likely sequence of states S_1^T, given a model λ and a sequence of observations Y_1^T. Since this task is performed after *all* of the observations in the sequence are available, there is more information available to perform it compared to the online recognition task. Several different algorithms have been used in order to perform this type of task, but the most commonly used is the Viterbi algorithm [14].

The Viterbi algorithm defines two matrices, $\delta(i,t)$ and $\psi(i,t)$. $\delta(i,t)$ represents "best path" probabilities for traveling particular paths in the sequence of states. For example, $\delta(3,4)$ is the probability of reaching state 3 at time 4 using the best or most likely path to get there. $\psi(i,t)$, on the other hand, represents "back pointers" to the states that produce the most probable paths. For example, if $\psi(3,4) = 2$, this means that, in order for the HMM to be in state 3 at time 4 with maximum probability $\delta(3,4)$, it must be in state 2 at time 3. Using the same notation $\delta(i,T)$ represents the probability of being in state i at the end of the sequence (i.e., time T).

The algorithm starts at time epoch one and then recursively calculates the successive values in $\delta(i,t)$ and $\psi(i,t)$. When the values of $\delta(i,T)$ have been calculated, the algorithm simply chooses the value of i that gives the maximum $\delta(i,T)$ and then backtracks through the sequence using values of $\psi(i,t)$ to determine the most likely path.

Formally, the Viterbi algorithm is implemented in four steps: 1) initialization, 2) recursion, 3) termination, and 4) path backtracking [17], described as follows.

1) Initialization: The values of $\delta(i,t)$ and $\psi(i,t)$ for time t=1 are set:

$$\begin{matrix} \delta(i,1) = \pi_i b_j(y_1) \\ \psi(i,1) = 0 \end{matrix} \quad 1 \le i \le M \tag{9.12}$$

Note that $\psi(i,1)$ has no clear meaning, since it specifies a state of the system prior to the first epoch. By convention it is set to zero, but it is never actually used in the algorithm.

2) Recursion: The values of $\delta(j,t)$ and $\psi(j,t)$ for times $t=2,\ldots,T$ are recursively calculated:

$$\begin{aligned} \delta(j,t) &= \max_{1\le i\le M}\left[\delta(i,t-1)a_{ij}\right]b_j(Y(t)) \\ \psi(j,t) &= \arg\max_{1\le i\le M}\left[\delta(i,t-1)a_{ij}\right] \end{aligned} \qquad \begin{cases} 2\le t\le T \\ 1\le j\le M \end{cases} \tag{9.13}$$

Note that we have changed the state index variable used in δ and ψ from i to j. This is done in order to remain consistent with the notation for a_{ij}, representing the probability of transitioning from state i to state j. Therefore, here i represents the state at time $t-1$ and j represents the state at time t.

3) Termination: Select the maximum probability, P_T, among the values of $\delta(i,T)$ and the corresponding state, $S(T)$, according to the following criteria:

$$\begin{aligned} P_T &= \max_{1\le i\le M}\left[\delta(i,T)\right] \\ S(T) &= \arg\max_{1\le i\le M}\left[\delta(i,T)\right] \end{aligned} \tag{9.14}$$

4) Path backtracking: Determine the corresponding most likely path (state sequence) using the back pointers, $\psi(i,t)$, and the following backward recursion:

$$S(t)=\psi(S(t+1),t+1) \quad t=T-1,T-2,\ldots,1 \tag{9.15}$$

9.3.3.4 Concluding Remarks about HMMs

This concludes our coverage of HMMs in this chapter. For readers not well-versed in probability theory (especially concepts such as Markov chains), these sections may have been challenging to digest. Fortunately, implementations of HMMs and the related algorithms for working with them are readily available in many programming languages (e.g., Java, C, and Matlab). In order to use them correctly, it is necessary to understand the concepts outlined here at a basic level. We have not addressed in this chapter various implementation details, such as techniques to avoid numeric underflow or variations of the algorithms to reduce memory requirements. Therefore, particular implementations may differ somewhat from what was presented here. More detailed coverage of HMMs can be found in [5, 18, 19].

9.3.4 The Sliding Window Method

As mentioned in Section 9.2.1, most of the techniques in machine learning have been developed to deal with *independent and identically distributed* (*iid*) data (HMMs are an exception to this). Such methods, in their original form, do not exploit the time dependence inherent in sequential data. A simple but effective technique to benefit from the wide array of available machine learning techniques yet also exploit this time dependency is the *sliding window method*, illustrated in Figure 9.5 [20]. This method effectively encapsulates the (local) time dependence of the data into a new data structure, a "sliding window," allowing the time-series data to be used in the same way as one would use iid data. As a result, one can use any of the classical (iid) machine learning techniques to classify samples of this sliding window.

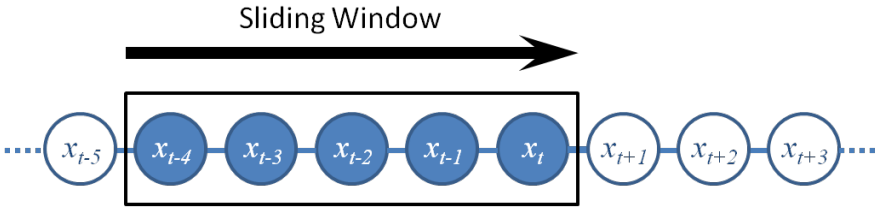


Figure 9.5 The sliding window method.

The best way to elucidate the sliding window method is by example. Suppose we have the following one-dimensional (unitless) data sequence:

$$X_1^{10} = (7,10,3,9,6,5,8,10,5,3)$$

First, a window width, w, must be chosen. The best choice of window width is dependent on the application and determines how much of the local time dependence is preserved in the data. Large windows will capture more time dependence but will also result in more computationally complex learning and classification. Therefore, the best choice is a trade-off between classification performance and computational time. In our example, we will choose a window size of five epochs.

Next, the windows are composed, which will result in a new data structure, H_t. This structure will have $w*d$ dimensions, where d is the original dimensionality of the data. In our case, H_t will be five-dimensional. There are two slightly different approaches to composing the windows, depending on how one wants to handle epochs 1 to $w - 1$, where there is not enough data to "fill" the window. The first approach is keep the number of samples in H_t the same as that of the original data and to use null values where necessary. For example, the first three windows composed from X_1^{10} would be:

$$H_1 = (\text{null}, \text{null}, \text{null}, \text{null}, 7)$$
$$H_2 = (\text{null}, \text{null}, \text{null}, 7, 10)$$
$$H_3 = (\text{null}, \text{null}, 7, 10, 3)$$

The other approach is simply to reduce the *length* of H_t by $w - 1$. Thus, epoch five from our original dataset becomes epoch one of H_t, and the first three windows become:

$$H_1 = (7, 10, 3, 9, 6)$$
$$H_2 = (10, 3, 9, 6, 5)$$
$$H_3 = (3, 9, 6, 5, 8)$$

In either case, as the window "slides" to the right in time, a new value from the sequence X_1^{10} comes in at the last (rightmost) position in H_t and an existing value (or null) slides out of the window with the values in between also being shifted accordingly.

After all the windows in H_t have been composed, the ordering of H_t is no longer important—it can now be regarded as a set instead of a sequence. In other words H_1, H_2, H_3, etc., can be shuffled like a deck of cards into any order. So long as the individual elements within each window are not rearranged, the local time dependence is preserved.

This is beneficial for many reasons. For example, in machine learning, data are usually separated into different groups for training the classifier and later for testing its performance. Now these groups can be chosen by randomly selecting samples from the set H_t, thus ensuring there is no selection bias in these groupings.

9.3.5 Bayesian Networks

Bayesian networks are directed graphical models, where nodes and directed links (i.e., edges) between the nodes represent conditional dependencies between variables (or sets of variables). As mentioned in Section 9.2.2, naïve Bayes' classifiers are a special case from the general framework of Bayesian networks. We saw in Section 9.2.2 that naïve Bayes' classifiers assume the input variables in the model are independent (i.e., no interconnecting arcs between them), but since this assumption is often not correct, Bayesian networks in general can model this interdependence among input variables. In fact, HMMs can also be viewed as a specific type of Bayesian network, called a *dynamic Bayesian network*, with certain constraints placed on its structure [23].

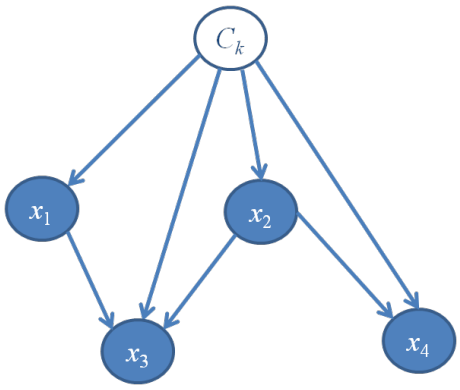


Figure 9.6 Example of a Bayesian network.

In general, Bayesian networks can be used to represent *any* joint probability distribution that consists of a product of conditional distributions. For example, the Bayesian network shown in Figure 9.6 corresponds to the joint distribution defined by the following equation:

$$p(C_k, x_1, x_2, x_3, x_4) = p(C_k)p(x_1 \mid C_k)p(x_2 \mid C_k)p(x_3 \mid C_k, x_1, x_2)p(x_4 \mid C_k, x_2) \tag{9.16}$$

In this way, Bayesian networks are a compact and intuitive means to express the probabilistic relationships among observable variables and hidden variables, such as the class label. Unfortunately, we don't have space to cover in detail how Bayesian networks are used in machine learning, but algorithms exist both to learn the graph structure from a labeled dataset and to perform inference on a given Bayesian network given unlabeled data. For detailed treatment of these subjects, see [5] or [24].

9.3.6 Decision Trees

Decision trees, or tree-based methods (which include several variants such as CART, ID3, and C4.5), comprise a simple but widely used machine learning technique that functions as a hierarchical set of if-then-else statements. For example, in the decision tree shown in Figure 9.7(a), classification is performed starting from the top node. For a particular data sample, if the condition lying between nodes 1 and 2 is satisfied (i.e., $x_1 > 5$), the classifier moves to the second node. Otherwise, it proceeds to node 3. The pattern is continued until a "leaf node" is reached (denoted in Figure 9.7 using uppercase letters), at which point a class label is assigned for that data sample.

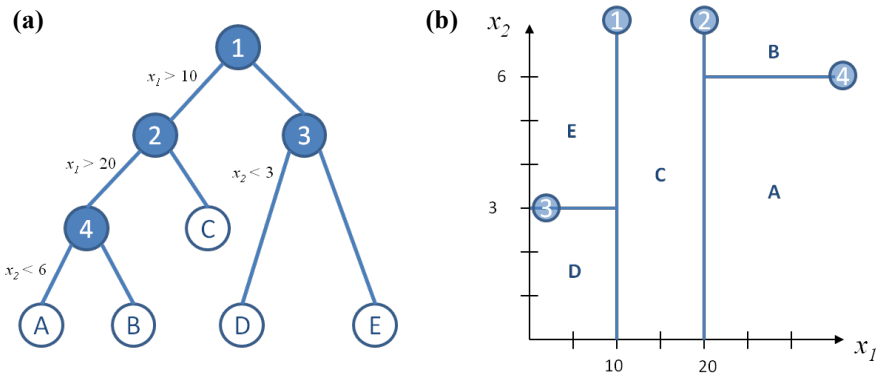


Figure 9.7 Example of a decision tree.

An alternative way to understand decision trees is that they divide an input space into distinct class regions, where each boundary corresponds to one node (i.e., "decision") from the decision tree, as shown in Figure 9.7(b). Each region can define a separate class (as in Figure 9.7), or several regions can be assigned to the same class. It is not necessary that the boundaries are orthogonal like in our example. In so-called oblique decision trees, linear combinations of variables can be used, in order to define non-orthogonal boundaries.

The overall goal with decision trees is to correctly classify as many data samples as possible, while minimizing the total number of regions (in order to avoid the problem of *overfitting*). There are two main alternatives for how to construct a decision tree. The first is to start with a single node and add one node at a time, until a threshold performance criteria is met. The other approach is to first grow a very large tree and then "prune" it (i.e., remove nodes), until a satisfactory balance is achieved between the performance of the tree and its complexity. A detailed discussion of decision trees can be found in [25].

9.3.7 Support Vector Machines (SVMs)

In recent years, SVMs have become a popular choice for building classifiers. The mathematics involved in constructing an SVM classifier are rather complex and beyond the scope of this book, but in this section we will provide sufficient detail to understand the basic concepts of SVMs.

9.3.7.1 Defining a Feature Space Using Kernel Functions

SVMs work by defining a much higher-dimensional space, called a *feature space*, mapped from the original input space. The added dimensions in the feature space are formed from transformations of one or more input variables [e.g., $\varphi: (x_i, x_j) \rightarrow$

$(x_i^2,\ x_i x_j,\ e^{x_j}, \ldots)]$. Generally speaking, it is not necessary to explicitly perform a mapping from the input space to the feature space, but rather the so-called *kernel trick* is used. The kernel trick uses a *kernel function*, $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, which corresponds to the inner product of the feature space. Here the indices i and j reference different data samples in the input space, which are assumed to be vectors. Common kernel functions used in SVMs include:

- Homogeneous polynomials: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i \cdot \boldsymbol{x}_j)^d$;
- Inhomogeneous polynomials: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i \cdot \boldsymbol{x}_j + c)^d$;
- Gaussian radial basis functions: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = exp\left(-\gamma \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|^2\right)$, where $\gamma > 0$;
- Hyperbolic tangents: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\kappa \boldsymbol{x}_i \cdot \boldsymbol{x}_j + c)$.

We will demonstrate the kernel trick with a simple example, where the input space is two-dimensional ($\mathbb{R}^2$). Here we use a homogeneous polynomial kernel function where d=2. We will compute the kernel function for two samples from the input space, and for greater clarity we define these as $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{y} = (y_1, y_2)$. Note that the subscripts now reference the dimension of the input space, as opposed to earlier where they referenced the different data sample. The kernel function for $\boldsymbol{x}$ and $\boldsymbol{y}$ is computed as follows:

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{y}) &= (\boldsymbol{x} \cdot \boldsymbol{y})^2 = (x_1 y_1 + x_2 y_2)^2 \\ &= x_1^2 y_1^2 + 2 x_1 y_1 x_2 y_2 + x_2^2 y_2^2 \end{aligned} \tag{9.17}$$

Inspecting (9.17), we can see that the corresponding feature space has three dimensions, and the two data samples in this feature space become:

$$\varphi(\boldsymbol{x}) = (x_1^2, \quad \sqrt{2} x_1 x_2, \quad x_2^2) \tag{9.18}$$

$$\varphi(\boldsymbol{y}) = \left(y_1^2, \quad \sqrt{2} y_1 y_2, \quad y_2^2\right) \tag{9.19}$$

We can verify (9.18) by checking that $k(\boldsymbol{x}, \boldsymbol{y}) = \varphi(\boldsymbol{x}) \cdot \varphi(\boldsymbol{y})$. As mentioned above, it is not necessary to explicitly map the input space to the feature space. Only the inner product, computed using the kernel function, is needed to build the classifier. In our simple example, it may seem like this is a trivial advantage, but in more complex examples, the feature space may consist of hundreds or thousands of dimensions. In fact, if a Gaussian radial basis function is used, it is usually evaluated using a Taylor series expansion. Therefore, the corresponding feature space would have infinite dimensions. Fortunately, only the inner product is needed, which can be truncated to desired precision by ignoring higher-order terms.

9.3.7.2 Maximum-Margin Hyperplanes

The main advantage of using a high-dimension space is that the data samples are more easily separated by *hyperplanes* (planes in spaces of arbitrary dimension), allowing linear classifiers to be defined in that hyperspace. These hyperplanes can be considered as similar to the boundaries between the class regions defined in decision trees (recall previous section). They are defined by maximizing the distance between the nearest data sample and the hyperplane, the so-called *maximum margin.* Such data samples lying nearest to the hyperplanes are called *support vectors* because their position supports the definition of the decision boundaries.

This concept is notionally depicted in Figure 9.8. The left side shows a two-dimensional space with data from two classes that are not linearly separable. On the right side, the data are remapped to a higher-dimensional space via a kernel function and plotted in only two selected dimensions (where the linear separation can be seen). The support vectors are the larger-sized data points lying closest to the solid red line, and this line constrains the orientation of the hyperplane in the high-dimensional space.

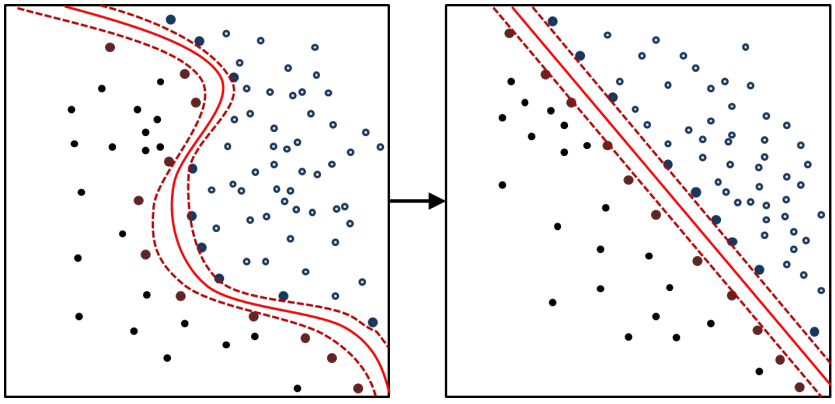


Figure 9.8 SVMs remap data into a higher-dimensional space, where the classes are linearly separable.

9.3.7.3 Concluding Remarks about SVMs

If one is interested in using SVMs to perform classification, it is not necessary to know the mathematical details about how to construct a maximum-margin hyperplane. Several open-source libraries exist for constructing SVMs and for using them to perform classification (e.g., [26, 27]). For most users of SVMs, it is probably sufficient to have a notional idea about how they function and perhaps

an understanding of how different kernel functions operate. Users can experiment with different kernel functions applied to their own datasets. The performance of the resulting SVM classifier will vary, depending on the dataset and the chosen kernel function. Because the kernel functions contain one or more free parameters, *parameter tuning* is often performed to optimize the performance. More detailed coverage of SVMs, including techniques for parameter tuning, can be found in [5, 14, 28].

9.4 SUMMARY

In this chapter, we defined contextual reasoning as *"the process of forming higher level inferences about context from lower level information."* We presented this process in the framework of the context pyramid, which aims to aid understanding of contextual reasoning at the conceptual level. We also provided a hypothetical example of this process within the domain of geospatial computing in smartphones.

Next we presented the primary method of contextual reasoning, namely machine learning. We presented three machine learning techniques in detail, including the naïve Bayes' classifier, HMMs, and the sliding window method. Finally, we gave a brief overview of several other commonly used machine learning techniques, including Bayesian networks, decision trees, and SVMs. Detailed coverage of these techniques can be found in the cited references.

In conclusion, by combining the wide variety of sensor data and geospatial information available in smartphones with state-of-the-art techniques of machine learning, smartphones can be made to "reason" about the context in which their users are living. This reasoning allows smartphones to become context-aware, enabling a wide range of context-aware applications and services.

References

[1] Puccio, G. J., M. C. Murdock, and M. Mance, *Creative Leadership: Skills That Drive Change*, San Diego, CA: Sage, 2007.

[2] Hurdus, J. G., and D. W. Hong, "Behavioral programming with hierarchy and parallelism in the DARPA urban challenge and RoboCup," *Proceedings of IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 20-22 Aug. 2008, 2008, pp. 503–509.

[3] Giunchiglia, F., "Contextual reasoning," *Epistemologia*, Special Issue on "I Linguaggi e le Macchine," Vol. XVI, 1993, pp. 345-364.

[4] Mitchell, T., et al., "Machine Learning," *Annual Review of Computer Science,* Vol. 4, 1990, pp. 417-433.

[5] Bishop, C. M., *Pattern Recognition and Machine Learning*, New York: Springer, 2006.

[6] O'Connor, B., "Statistics vs. machine learning, fight!" *AI and Social Science* (blog), http://bit.ly/VtpHEX, 2008.

[7] Shumway, R. H., and D. S. Stoffer, *Time Series Analysis and Its Applications: With R Examples*, New York: Springer, 2011.

[8] Kingsbury, N., D. B. H. Tayy, and M. Palaniswamiz, "Multi-scale kernel methods for classification," *Proceedings of the 2005 IEEE Workshop on Machine Learning for Signal Processing*, 2005, pp. 43-48.

[9] Visatemongkolchai, A., and H. Zhang, "Building probabilistic motion models for SLAM," *Proceedings of the 2007 IEEE International Conference on Robotics and Biomimetics*, 2007, pp.1629-1634.

[10] Ting, S. L., W. H. Ip, and A. H. C. Tsang, "Is naïve Bayes a good classifier for document classification?" *International Journal of Software Engineering and Its Applications*, Vol. 5, No. 3, 2011, pp. 37-46.

[11] Boiman, O., E. Shechtman, and M. Irani, "In defense of nearest-neighbor based image classification," *IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1-8.

[12] Tapia, E. M., S. S. Intille, and K. Larson, "Activity recognition in the home setting using simple and ubiquitous sensors," *Pervasive Computing: Second International Conference, PERVASIVE 2004*, pp. 158-175, A. Ferscha and F. Mattern (eds.), Berlin: Springer, 2004.

[13] Bancroft, J. B., D. Garrett, and G. Lachapelle, "Activity and environment classification using foot mounted navigation sensors," *2012 International Conference on Indoor Positioning and Indoor Navigation*, 2012.

[14] Alpaydin, E., *Introduction to Machine Learning, Second Edition*, Cambridge, MA: The MIT Press, 2010.

[15] Bordes, L., and P. Vandekerkhove, "Statistical inference for partially hidden Markov models," *Communications in Statistics—Theory and Methods*, Vol. 34, No. 5, 2005, pp. 1081-1104.

[16] Ristic, B., S. Arulampalm, and N. Gordon, *Beyond the Kalman Filter: Particle Filters for Tracking Applications*, Norwood, MA: Artech House, 2004.

[17] Liu, J., "Hybrid Positioning with Smart Phones," In *Ubiquitous Positioning and Mobile Location-Based Services in Smart Phones*, pp. 159-193, R. Chen (ed.), Hershey, PA: IGI-Global, 2012.

[18] Rabiner, L. R., "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, Vol. 77, No. 2, 1989, pp. 257-286.

[19] Fraser, A. M., *Hidden Markov Models and Dynamical Systems,* Philadelphia, PA: Society for Industrial and Applied Mathematics, 2008.

[20] Dietterich, T. G., "Machine learning for sequential data: A review," *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, 2002, pp. 15-30.

[21] Wolpert, D., "The lack of a priori distinctions between learning algorithms," *Neural Computation*, Vol. 8, No. 7, 1996, pp. 1341-1390.

[22] Schaffer, C., "A conservation law for generalization performance," *Proceedings of the Eleventh International Conference on Machine Learning*, 1994, pp. 259-265.

[23] Ghahramani, Z., "An introduction to hidden Markov models and Bayesian networks," *International Journal of Pattern Recognition and Artificial Intelligence*, Vol. 15, No. 1, 2001, pp. 9-42.

[24] Friedman, N., D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine Learning*, Vol. 29, 1997, pp. 131-163.

[25] Sutton, C. D., "Classification and Regression Trees, Bagging, and Boosting," *Handbook of Statistics*, Vol. 24, 2005, 303-329.

[26] Chang, C. C., and C. J. Lin, *LIBSVM—A Library for Support Vector Machines*, http://bit.ly/VkqyrS, 2012.

[27] Katholieke Universiteit Leuven, *Least Squares—Support Vector Machines,* http://bit.ly/VkurwO, 2012.

[28] Steinwart, I., and A. Christmann, *Support Vector Machines*, New York: Springer, 2008.

[29] Sutton, C., and A. McCallum, "An Introduction to Conditional Random Fields," *Foundations and Trends in Machine Learning*, Vol. 4, No. 4, 2011, 267–373.

[30] Lafferty, J. D., A. McCallum, F. C. N. Pereira, "Conditional random fields: probabilistic models for segmenting and labeling sequence data," *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001, pp. 282-289.

[31] Klinger, R., and K. Tomanek, *Classical Probabilistic Models and Conditional Random Fields*, Algorithm Engineering Report TR07-2-013, Technical University of Dortmund, 2007.

[32] Van Kasteren, T., A. Noulas, G. Englebienne and B. Kröse, "Accurate Activity Recognition in a Home Setting", In *Proceedings of the 10th international conference on Ubiquitous computing (UbiComp '08)*, pp. 1-9, 2008.

[33] Fisher, R.A, The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[34] Abidine, M. B., "Evaluating C-SVM, CRF and LDA classification for daily activity recognition," *Multimedia Computing and Systems (ICMCS), 2012 International Conference on*, 2012, pp. 272–277.

[35] Ward, J. A., et al., "Activity recognition of assembly tasks using body-worn microphones and accelerometers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 28, No. 10, 2006, pp. 1553-1567.

[36] Andreu, J., R. D. Baruah, and P. Angelov, "Real Time Recognition of Human Activities from Wearable Sensors by Evolving Classifiers," *2011 IEEE International Conference on Fuzzy Systems*, 2011, pp. 2786-2793.

[37] Phung, D., et al., "High accuracy context recovery using clustering mechanisms," *IEEE International Conference on Pervasive Computing and Communications, (PerCom)2009*, 2009, pp. 1-9.

[38] Aziz, O., and S. N. Robinovitch, "An analysis of the accuracy of wearable sensors for classifying the causes of falls in humans," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, Vol. 19, No. 6, 2011, pp. 670-676.

[39] Javed, J., H. Yasin, and S. F. Ali, "Human movement recognition using Euclidean Distance: A tricky approach," *Image and Signal Processing (CISP), 2010 3rd International Congress on*, 2010, pp. 317-321.

[40] Donohoo, B., et al., "Exploiting spatiotemporal and device contexts for energy-efficient mobile embedded systems," *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 1274-1279.

[41] Krishnan, N. C., and S. Panchanathan, "Analysis of low resolution accelerometer data for continuous human activity recognition," *Acoustics, Speech and Signal Processing (ICASSP) 2008. IEEE International Conference on*, 2008, pp. 3337-3340.

[42] Kwapisz, J. R., G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," *ACM SIGKDD Explorations Newsletter,* Vol. 12, No. 2, 2011, pp. 74-82.

[43] Vail, D. L., M. M. Veloso, and J. D. Lafferty, "Conditional random fields for activity recognition," *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2007, pp. 1331-1338.

[44] Kohonen, T., "The self-organizing map," *Proceedings of the IEEE* 78, no. 9, 1990: 1464-1480.

[45] Suzuki, S., et al., "Activity recognition for children using self-organizing map," *RO-MAN, 2012 IEEE*, 2012, pp. 653-658.

[46] Huang, W., and J. Wu, "Human action recognition based on self organizing map," *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2010, pp. 2130-2133.