7 Drawing It All Together: Two Examples

This chapter draws together the entire material presented so far in two detailed examples. The first example involves the WITNESS model of eyewitness identification (Clark, 2003) and in particular its application to the "verbal overshadowing effect" reported by Clare and Lewandowsky (2004). The second example involves a head-to-head comparison of some models of categorization, thereby illustrating the concepts of model selection developed in Chapter 5.

These two examples illustrate several important contrasts: First, WITNESS is based on a stochastic simulation involving a large number of replications, whereas the categorization models are based on analytic solutions and hence provide predictions that are not subject to sampling variability. Second, WITNESS considers the data at the aggregate level because each subject in an eyewitness identification experiment provides only a single response, and response proportions are thus only available in the aggregate, whereas the categorization models can be fit to the data from individuals. Third, the WITNESS example is based on a descriptive approach relying on a least squares criterion (i.e., root mean squared deviation [RMSD]; see Chapter 3), whereas the comparison of the categorization models involves maximum likelihood estimation and model comparison (see Chapters 4 and 5).

7.1 WITNESS: Simulating Eyewitness Identification

In 1979, a Catholic priest, Father Bernard Pagano, was on trial in New Jersey on multiple charges of armed robbery. The prosecution's case was supported by the fact that the defendant had been positively identified by seven (7!) independent eyewitnesses. After the prosecution had presented its case, the actual perpetrator, a Mr. Ronald Clouser, came forward and confessed to the crimes after having been identified and located by a former FBI agent. The charges against Father Pagano were dropped (see Searleman & Herrmann, 1994, for a recounting of this intriguing case). How could so many eyewitnesses have mistakenly identified an innocent person? Lest one dismiss this as an isolated case, Wells et al. (1998) presented a sample of 40 cases in which wrongfully convicted defendants were exonerated on the basis of newly available DNA evidence. In 90% of those cases, eyewitness identification evidence was involved—in one case, a person was erroneously identified by five different witnesses.

Not surprisingly, therefore, the study of eyewitness behavior has attracted considerable attention during the past few decades. In eyewitness identification experiments, subjects typically watch a (staged) crime and are then presented with a lineup consisting of a number of photographs of people (for an overview, see Wells & Seelau, 1995). The lineup typically contains one individual who committed the (staged) crime, known as the perpetrator, and a number of others who had nothing to do with the crime, known as foils. Occasionally, "blank" lineups may be presented that consist only of foils. There is little doubt that people's performance in these experiments is far from accurate, with false identification rates—that is, identification of a foil—in excess of 70% (e.g., Clare & Lewandowsky, 2004; Wells, 1993). The laboratory results thus confirm and underscore the known problems with real-world identification evidence.

The first computational model of eyewitness identification, appropriately called WITNESS, was proposed by Clark (2003). WITNESS provided the first detailed description of the behavior of eyewitnesses when confronted with a police lineup or its laboratory equivalent, and it has successfully accounted for several diagnostic results (Clark, 2003, 2008).

7.1.1 WITNESS: Architecture

WITNESS belongs to the class of direct-access matching models of memory (Clark & Gronlund, 1996), in which recognition decisions are based on direct comparisons between the test items—that is, the people in the lineup—and the contents of memory. In WITNESS, the only relevant content of memory is assumed to be the face of the perpetrator. WITNESS is based on the following architectural principles:

(1) All stimuli are represented as random vectors ($f_1, f_2, \ldots, f_k$) with features drawn from a uniform distribution with mean zero (and range $-.5$ to $+.5$). One of those vectors represents the perpetrator, whereas the others represent the foils in the lineup.

(2) Encoding into memory is assumed to be imperfect, such that only a proportion s ($0 < s < 1$) of features are veridically copied into a memory vector (called $\boldsymbol{M}$) when the perpetrator is witnessed during commission of the crime. The value of s is typically estimated from the data as a free parameter.[1] The remaining $1 - s$ features are stored incorrectly and hence are replaced in memory by another sample from the uniform distribution.

(3) At the heart of WITNESS's explanatory power is its ability to handle rather complex similarity relationships between the perpetrator and the foils in a lineup, which vary with the way foils are selected (see Clark, 2003, for details). For the present example, we simplified this structure to be captured by a single parameter, sim, which determined the proportion of features ($0 < sim < 1$) that were identical between any two vectors, with the remainder ($1 - sim$) being randomly chosen from the same uniform distribution (range $-.5$ to $+.5$). Thus, all foils in the lineup resembled the perpetrator to the extent determined by the sim parameter.

(4) At retrieval, all faces in the lineup are compared to memory by computing the dot product between the vector representing each face and $\boldsymbol{M}$. The dot product, d, is a measure of similarity between two vectors and is computed as

$$d = \sum_{i=1}^{N} g_i M_i, \tag{7.1}$$

where N is the number of features in each vector and i the subscript running over those features. The greater the dot product, the greater the similarity between the two vectors. In WITNESS, the recognition decision relies on evaluating the set of dot products between the faces in the lineup and $\boldsymbol{M}$.

(5) The complete WITNESS model differentiates between three response types: an identification of a lineup member ("it's him"), a rejection of the entire lineup ("the perpetrator is not present"), and a "don't know" response that was made when there was insufficient evidence for an identification (Clark, 2003). For the present example, we simplified this decision rule by eliminating the "don't know" category in accordance with the experimental method to which we applied the model. This simplified decision rule reduces to a single comparison: If the best match between a lineup member and memory exceeds a criterion, c_{rec}, the model chooses that best match as its response. If *all* matches fell below c_{rec}, the model rejects the lineup and records a "not present" response.

Because each participant in an eyewitness identification experiment provides only a single response, the data are best considered in the aggregate. In particular, the data consist of the proportion of participants in a given condition who identify the perpetrator or one of the foils, or say "not present." The model predictions are likewise generated by aggregating across numerous replications, each of which involved a different set of randomly constructed stimulus vectors. Each replication can be taken to represent a unique participant who catches a uniquely encoded glimpse of the perpetrator.[2]

7.1.2 WITNESS and Verbal Overshadowing

A common task of eyewitnesses is to provide police with a verbal description of the perpetrator, in the hope that this may lead to the apprehension of a suspect for subsequent identification from a lineup. Although providing a verbal description is standard police procedure, Schooler and Engstler-Schooler (1990) reported an unexpected adverse side effect of verbal descriptions. In their study, subjects viewed a staged crime and then either provided a verbal description of the perpetrator (verbalization condition) or completed an irrelevant filler task (control condition). Following this manipulation, witnesses attempted to identify the perpetrator from a photo lineup. The verbalization condition yielded significantly fewer correct identifications than the control condition. Schooler and Engstler-Schooler termed this adverse influence of verbalization on identification the *verbal overshadowing* effect. This initial study was followed by much research activity, and a meta-analysis of a large number of published and unpublished studies by Meissner and Brigham (2001) confirmed the presence of a small, but significant, negative effect of verbalization on identification accuracy. How might one explain the verbal overshadowing effect? Why would verbalization disrupt one's visual memory of a person's face?

There are several candidate explanations for verbal overshadowing, and the example below is using the WITNESS model to differentiate between them. We focus on two candidates, which for brevity we refer to as the *memory* and *criterion* explanation, respectively. According to the memory explanation, the verbalization harms and disrupts one's memory for the perpetrator, for example, when the verbal description inadvertently entails generation of incorrect elements. According to the criterion explanation, by contrast, verbalization does not alter the memory of the perpetrator but adjusts one's response criterion upward. That is, owing to the subjective difficulty most people experience during an attempt to describe a person, they become more reluctant to choose someone subsequently from the lineup—thus giving rise to the *appearance* of impaired recognition accuracy.

The two explanations can be empirically differentiated by manipulating the type of lineup and the decision required of subjects: First, in a forced-choice situation, in which people *must* identify someone from the lineup, the criterion explanation predicts that no verbal overshadowing should occur. If a choice is mandatory, then a response criterion does not matter. The memory explanation, by contrast, predicts the presence of verbal overshadowing even with forced-choice lineups.

Second, in an optional-choice situation, in which people may make a "not there" response, both explanations expect verbal overshadowing to be present; however, the criterion explanation expects the effect to arise from an increase in (erroneous) "not there" responses, whereas the memory explanation expects the effect to involve primarily an increase in false identifications (of a foil) from the lineup.

Third, if the perpetrator is not present in an optional-choice lineup (which corresponds to a situation in which the police have apprehended the wrong person), then the criterion explanation predicts an *increase* in accuracy after verbalization. If verbalization renders people more conservative, thus making an identification less likely, accuracy in perpetrator-absent lineups should be enhanced. The memory explanation, by contrast, expects overshadowing to have a detrimental effect even with perpetrator-absent lineups because an impaired memory should impair accuracy irrespective of whether or not the perpetrator is presented.

Clare and Lewandowsky (2004) reported three experiments that sought to differentiate between those explanations. The results of all studies clearly confirmed the predictions of the criterion explanation and, by implication, rejected the memory explanation. We focus on two of their studies (their Experiments 1 and 2) that involved an optional-choice lineup in which the perpetrator either was or was not present (Experiment 1) and a forced-choice lineup in which the perpetrator was always present (Experiment 2). Table 7.1 summarizes the results from those two studies.

In each experiment, there were three conditions: a control condition in which people had to perform a verbal task that was unrelated to eyewitness identification and two verbalization conditions that differed only with respect to what people were to emphasize in their descriptions—namely, holistic aspects of the face (i.e., traits such as intelligence, friendliness, etc.) or featural aspects (i.e., hair color, shape of nose, skin tone, etc.). In Experiment 2, in which an identification was mandated, identification rates did not differ between the three conditions, exactly as predicted by the criterion explanation. In Experiment 1, where an identification was optional, the verbal description conditions differed considerably from the control condition, but the direction of the effect was reversed with lineup type. When the perpetrator was present, people were less accurate after providing a description, whereas when the perpetrator was absent, people were more accurate—but in both instances, this change in accuracy reflected a common tendency to be less likely to identify someone (and, correspondingly, to be more likely to respond "not there") after verbalization.

The data clearly seem to favor the criterion explanation. However, there are at least two reasons why modeling is advisable before the explanation can be accepted: First, we must demonstrate that the explanation *can* be instantiated in a model and that the model can quantitatively handle the data (recall the surprising

Table 7.1 Eyewitness Responses in Two Experiments Reported by Clare and Lewandowsky (2004)

			Verbalization Condition		
Lineup[a]	Decision	Response Type[b]	Control	Holistic	Featural
		Experiment 1			
PP	Optional choice	Hit	.80	.57	.69
		False ID	.13	.06	.12
		Miss	.07	.36	.19
PA	Optional choice	CR	.23	.52	.52
		False ID[c]	.77	.48	.48
		Suspect	.04	.20	.00
		Foil	.73	.28	.48
		Experiment 2			
PP	Forced choice	Hit	.86	.81	.84
		False ID	.14	.19	.16

a. PP = perpetrator present; PA = perpetrator absent.
b. Hit = correct identification; False ID = identification of foil; Miss = erroneous "not there" response; CR = correct rejection.
c. False IDs with the perpetrator-absent lineup are further broken down by "suspect" versus the other foils.

difficulties we faced in Chapter 2 when we tried to instantiate the phonological loop model). Second, we must rule out the possibility that some variant of the memory explanation might handle the data after all. We now present the modeling that resolved both of those issues.

7.1.3 WITNESS in MATLAB

Clare and Lewandowsky (2004) applied WITNESS to the data from all three of their experiments using six free parameters. For this example, we focus on their Experiments 1 and 2 and rely on a slightly simpler five-parameter version of WITNESS that was fit to both studies simultaneously. We begin by presenting the criterion explanation within WITNESS.

Table 7.2 summarizes the parameters and their best-fitting estimates (which you will be able to reproduce by the time you have worked through the MATLAB programs that we discuss next).

The first three parameters are exactly as discussed in Section 7.1.1. The last two parameters, $c_{rec}(H)$ and $c_{rec}(F)$, instantiate the criterion explanation and

Table 7.2 Free Parameters in WITNESS

Parameter		Best-Fitting Estimate
Encoding strength	s	.27
Similarity	sim	.29
Baseline criterion	$c_{rec}(C)$	1.20
Holistic criterion	$c_{rec}(H)$	1.84
Featural criterion	$c_{rec}(F)$	1.64

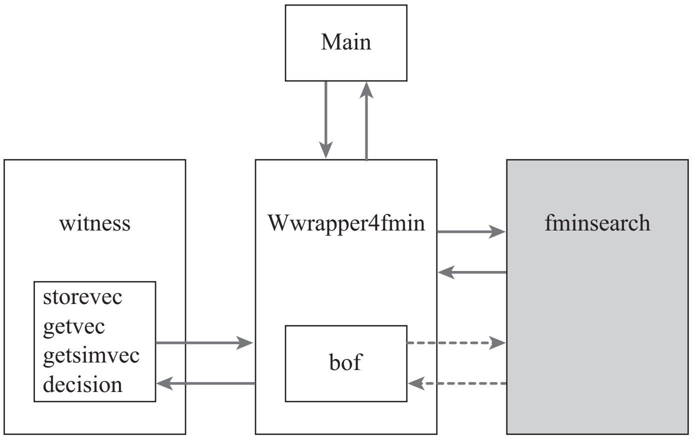

Figure 7.1 The relationship between the MATLAB functions used in the WITNESS simulation. The names in each box refer to the function name(s) and file names. Boxes within a box represent embedded functions. Arrows refer to exchanges of information (via function calls and returns or global variables). Solid arrows represent information exchanges that are managed by the programmer, whereas broken arrows represent exchanges managed by MATLAB. Shading of a box indicates that the function is provided by MATLAB and does not need to be programmed. See text for details.

represent, respectively, the value of the response criterion after holistic and featural verbalization. Aside from altering the setting of the criterion, verbalization has no further effect within this version of WITNESS.[3]

7.1.3.1 The Calling Sequence

Our example adheres to the schema introduced in Chapter 3 (Figure 3.2). To guide our presentation, we show the relationship among our various programs in Figure 7.1. The figure is nearly identical to the earlier schema (Figure 3.2) except that the names of the functions have changed to reflect the current situation.

We now present and discuss the various programs, beginning with the function that implements WITNESS and that is called from within `bof`, the function that computes badness of fit. We present the main program and the `Wwrapper4fmin` function later.

7.1.3.2 *The WITNESS Function*

We begin by presenting in Listing 7.1 the core of the function that implements WITNESS. The function takes a single input argument that contains the current parameter values, and it returns the associated predictions in a single array. In addition, the function uses a "global" variable—defined immediately below the function header—to communicate with other MATLAB programs and functions that are part of the simulation package. Variables that are declared global in MATLAB are accessible from within all functions in which they appear in a global statement; this provides another convenient avenue of communication between parts of a large program that does not require input or output arguments.

Listing 7.1 The WITNESS Function

```
function predictions = witness (parms)
% Implementation of the WITNESS model for
% "Computational Modeling in Cognition: Principles ↩
    and Practice"

global consts;

rand ('state', consts.seed)

s    = parms(1);
sim  = parms(2);
ssp  = sim;
paSim= sim;
ppSim= sim;

predictions = zeros (consts.nCond, 3);
for reps=1:consts.nRep
    %obtain perpetrator and perform holdup
    perp=getvec(consts.n);
    m=storevec (s, perp);

    %get an innocent suspect
    inSus=getsimvec(ssp, perp);
    %create both types of lineup
    paLineup (1,:)= inSus;
    ppLineup (1,:)= perp;
```

```
26      for i=2:consts.lSize
27          paLineup (i,:) =  getsimvec (paSim, perp);
28          ppLineup (i,:) =  getsimvec (ppSim, perp);
29      end
30
31      %eyewitness inspects lineup
32      for i=1:consts.lSize
33          paMatch(i) = dot(paLineup(i,:) , m);
34          ppMatch(i) = dot(ppLineup(i,:) , m);
35      end
36
37      %witness responds
38      for iLineup=1:consts.nCond
39          if any(iLineup==consts.fChoice)
40              criterion=0;
41          else
42             criterion=parms(consts.ptToCrit(iLineup));
43          end
44          if any(iLineup==consts.paLineup)
45              useMatch = paMatch;
46          else
47              useMatch = ppMatch;
48          end
49          resp = decision (useMatch, criterion);
50          predictions (iLineup, resp) = ↩
                predictions(iLineup, resp) + 1;
51      end
52  end %rep loop
```

Preliminaries. First consider the statement in line 7, which calls the random-number generator with two arguments that reset its state to the value provided by the variable `consts.seed`.[4] Note that this usage of `consts.seed` identifies the global variable `consts` to be a "structure." A structure is a very useful construct in MATLAB because it allows you to refer to many variables (known as "structure members") at the same time, in much the same way that checking in a single suitcase is far preferable to carrying numerous socks and shirts to the airport. Structure members are identified by appending their names to the structure's name, separated by a period (".").

Because the `consts` structure contains several important variables that govern the simulation, we summarize all its members and their values in Table 7.3. We will show later how those structure members are initialized; for now, we can take their values for granted as shown in the table. The structure members that are identified by an asterisk will be explained further in the following; the others are self-explanatory.

Table 7.3 Members of the `consts` Structure in WITNESS

Member Name	Explanation	Value
`consts.seed`	Seed for random generator	21335
`consts.lSize`	Size of lineup	6
`consts.nRep`	Number of simulation replications	1000
`consts.n`	Number of features in vectors	100
`consts.nCond`	Number of conditions modeled	7
`consts.fChoice`	Forced-choice conditions*	[7 8 9]
`consts.paLineup`	Conditions without perpetrator*	[4 5 6]
`consts.ptToCrit`	Pointer to appropriate criterion*	[3 4 5 3 4 5]
`consts.maxParms`	Maximums for parameters*	[1 1 inf inf inf]

The reseeding of the random generator ensures that the sequence of random numbers provided by MATLAB's rand function is identical every time the `witness` function is called. In consequence, there are no uncontrolled random variations across calls of the function during parameter estimation, thus minimizing the disturbance of the error surface that is associated with stochastic simulation models (see Chapter 3).[5]

The subsequent lines (9–13) assign the first two entries in the parameter vector to more mnemonic variable names. Thus, we create a variable `s` that contains the current value of the encoding parameter, s, and a variable `sim` for the parameter of the same name. (In case you are wondering how we know that those two parameters take the first two slots in the parameter vector, the answer is that the order of parameters in the vector corresponds to their order of presentation in Table 7.2; this order was determined by us, the programmers.) We then assign the same value of `sim` to three other variables that represent the specific similarities within the simulation—namely, between the perpetrator and an innocent suspect (i.e., the person that takes the perpetrator's place on perpetrator-absent lineups; variable `ssp`) and between the perpetrator and all foils on the perpetrator-present (`ppSim`) and perpetrator-absent (`paSim`) lineups. The reason we use different variable names here even though all are assigned the same value is to leave open the possibility that in future simulations, the different similarities may take on distinct values.

The core components. The core of the function begins in line 16 with a loop that accumulates predictions across the multiple replications. Within the loop, each replication first involves a holdup (or some other heinous crime), which is modeled in line 19 by storing in memory an image of the perpetrator (generated in the line immediately above). The two functions `getvec` and `storevec` form part of the WITNESS simulation package (as foreshadowed in Figure 7.1) and are presented later; for now, it suffices to know that they generate a random vector and store a vector in memory, respectively.

The holdup is immediately followed by lineup construction. First, an innocent suspect is obtained for the perpetrator-absent lineup using another embedded function—namely, `getsimvec` (Line 22). The two lineup types are then created by placing the perpetrator or innocent suspect in the first lineup position (lines 24 and 25) and the foils in the remaining positions (lines 26–29). At this point, your training in experimental design should kick in, and you should balk at the idea that the first lineup position is always taken up by the perpetrator (when present). Surely this important variable must be randomized or counterbalanced? Yes, if this were a behavioral experiment involving human subjects whose decisions are subject to all sorts of biases, then it would be imperative to determine the position of the perpetrator at random. Fortunately, however, WITNESS does not suffer from such biases and considers all lineup positions exactly equally. For that reason, we can fix the perpetrator's position, which turns out greatly to facilitate scoring.

You might also wonder why we created two separate lineups, rather than just a single set of foils that is presented either with the perpetrator or an innocent suspect in the first position. Does the use of different foils for the perpetrator-present and perpetrator-absent lineups not introduce an unnecessary source of variation? The answer is that in reality, foils are selected by police officers in order to match the apprehended person of interest—who may or may not be the perpetrator (for details, see Clark & Tunnicliff, 2001). It follows that in order to be realistic, the foils should differ between the two lineup types. However, because `ppSim` and `paSim` are set to the same value, there should be no systematic differences between foils in the two lineup types.

Once created, the witness inspects the lineups, and a match between memory and each lineup member is computed (lines 32–35). In contrast to human witnesses, who can only be assigned to one lineup type or the other—but not both—the model can consider two completely different conditions without any bias or carryover effects. This means that although it looks like we are testing the same witness under different conditions, we are actually testing different simulated participants in line with the experimental design. The match is computed by calling the MATLAB function dot, which returns the dot product between two vectors.

Response selection and scoring. WITNESS then selects a response in the manner dictated by the experimental methodology and in line with the criterion explanation. This crucial segment, which implements the experimental methodology underlying the data in Table 7.1, involves the loop beginning in line 38.

At this point things get to be somewhat intricate, and to facilitate understanding, we present the mapping between experimental conditions and the program variables in Figure 7.2.

Experiment	1						2		
Lineup	PP			PA			PP		
Condition	Control	Holistic	Featural	Control	Holistic	Featural	Control	Holistic	Featural
iLineup	1	2	3	4	5	6	7	8	9
consts.ptToCrit	3	4	5	3	4	5	n/a	n/a	n/a
criterion	parms (3)	parms (4)	parms (5)	parms (3)	parms (4)	parms (5)	0	0	0
any(iLineup==consts.paLineup)	0	0	0	1	1	1	0	0	0
any(iLineup==consts.fChoice)	0	0	0	0	0	0	1	1	1

Figure 7.2 Mapping between experimental conditions (shaded part at the top) and program parameters in our simulation (bottom part). PP = perpetrator-present lineup; PA = perpetrator-absent lineup. See text for details.

The shaded cells at the top of the figure summarize the data that we want to simulate: There are two experiments, there are two lineup types, and there are three conditions for each lineup type in each experiment. Now consider the bottom (unshaded) panel of the figure, which lists the values of the program variables that, in lines 38 through 51, instantiate this experimental setup, thus ensuring that WITNESS selects responses in the manner appropriate for the experiment, condition, and lineup type being modeled. Turning first to Experiment 2, the response criterion is disabled by setting it to zero (line 40) whenever the loop index `iLineup` matches one of the elements of `consts.fChoice`, which is an array that points to the conditions that comprise Experiment 2 (see Table 7.3 and Figure 7.2). Note that the criterion explanation cannot differentiate between the three conditions within a forced-choice methodology, which implies that to the extent that there are differences between the conditions in Experiment 2, this will necessarily increase the misfit of the model.

For the remaining optional-choice conditions from Experiment 1, two choices must be made: The appropriate criterion must be selected from the parameter vector, and the appropriate lineup must be chosen. The criterion is chosen in line 42 using the array `consts.ptToCrit`. The first three rows in the bottom panel of Figure 7.2 clarify the mapping between the loop index (`iLineup`) and the decision criterion that is being selected from the parameter vector. The lineup type is determined in line 44, using the array `consts.paLineup`. The second-last row in Figure 7.2 shows the outcome of the expression in line 44, which is equal to 1 (i.e., true) whenever the value of the loop index (`iLineup`) matches one of the perpetrator-absent conditions.

Finally, once the criterion and lineup type have been selected, a response is returned by the `decision` function, which we will discuss in a moment. The returned responses are counted in the `predictions` array (line 50), which keeps track of the number of occurrences of each response type for each condition.

To summarize, within each replication, WITNESS encounters a perpetrator and then selects a response under all conditions being modeled—that is, two lineup types (perpetrator present or absent), two decision types (optional choice or forced choice), and three experimental conditions (control, holistic, and featural). Once all replications are completed, the counted responses are converted to predicted proportions (not shown in listing) and are returned as the model's predictions.

You may have noted that a seemingly disproportionate amount of program (and description) space was devoted to vaguely annoying matters of bookkeeping, such as identification of the appropriate parameters and lineups for the various conditions. Comparatively little space seemed to be devoted to doing the actual simulation, for example, the encoding of the perpetrator's face. This is not at all unusual: In our experience, the majority of programming effort tends to be devoted to instantiating important details of the experimental method and to keeping track of simulation results.

Embedded functions. Let us now turn to the various embedded functions that are required to make WITNESS work. Listing 7.2 shows the remaining segment of the `witness` function; as indicated by the line numbers, this listing involves the same file shown above in Listing 7.1.

Listing 7.2 Embedded Functions for WITNESS

```
54
55
56 %------ miscellaneous embedded functions
57 %get random vector
58     function rv = getvec (n)
59         rv = (rand(1,n)-0.5);
60     end
61
62 %take a vector and return one of specified similarity
63     function outVec=getsimvec (s, inVec)
64         a = rand(1,length(inVec)) < s;
65         outVec = a.*inVec  + ~a.*getvec(length(inVec));
66     end
67
68 %encode a vector in memory
69     function m=storevec (s, inVec)
70         m = getsimvec(s, inVec);
71     end
72
73 %implement the decision rules
74     function resp = decision(matchValues, cRec)
```

(Continued)

(Continued)

```
75         %if all lineup members fall below cRec, then ↩
               reject
76         if max(matchValues) < cRec
77             resp=3;
78         else
79             [c,j]=max(matchValues);
80             if j == 1    %suspect or perp always ↩
                   first
81                 resp=1;
82             else
83                 resp=2;
84             end
85         end
86     end
87 end
```

The first embedded function in line 58, `getvec`, is simplicity itself: It creates a vector of uniform random numbers that are centered on zero and range from $-.5$ to $+.5$. All stimulus vectors in this simulation are ultimately generated by this function.

The next function, `getsimvec` in line 63, also returns a random vector, but in this instance, the new vector is of a specified similarity to another one that is provided by the input argument `inVec`. Specifically, the function returns a vector in which a random proportion s of features are drawn from `inVec`, and the remainder is sampled at random.

To encode the perpetrator in memory, we use the function `storevec` in line 69: As it turns out, this function simply calls `getsimvec`, thus instantiating WITNESS's assumption that only part of the perpetrator image is stored correctly, whereas the remaining encoded features are sampled at random. (In fact, we could have omitted this function altogether and used `getsimvec` to do the encoding. However, by using a separate function, we leave open the door for possible future modifications of the encoding process.)

Finally, we must examine how WITNESS selects a response. This selection is made by the function `decision`, which is defined in lines 74 through 86. The function receives an array of dot products that represent the match between memory and the lineup members (input argument `matchValues`) together with a response criterion (`cRec`). If all matches fall below the criterion (line 76), then a response type "3" is returned. Alternatively, the lineup member with the largest match is returned as the response. If that largest match is in Position 1 (line 80), then we know that we have identified the perpetrator (or innocent suspect, in perpetrator-absent lineups), and the function returns a response type "1." (Remember how we said earlier that placing the perpetrator in Position 1 facilitates scoring—now you know why.) Alternatively, if the largest match is in any other position, we return response type “2,” which means that a foil has been mistakenly identified. To summarize, the `decision` function returns a single variable that can take on values 1, 2, or 3 and classifies the response, respectively, as (1) an identification of the perpetrator, (2) an identification of a foil, or (3) the rejection of the lineup. Recall that those response types are counted separately across replications (refer back to line 50 in Listing 7.1).

This, then, completes the presentation and discussion of the central part of the simulation—namely, the `witness` function and all its embedded auxiliary functions. Within the structure in Figure 7.1, we have discussed the box on the left. We next turn to the main program shown in the box at the top of that figure.

7.1.3.3 The Main Program

The compact main program is presented in Listing 7.3 and is explained quite readily. Lines 6 through 13 initialize the `consts` structure with the values shown earlier in Table 7.3. Note how those values were available inside the `witness` function because `consts` was declared to be global both here and inside the function.

Listing 7.3 Main Program for WITNESS Simulation

```
 1 % Program to estimate parameters for WITNESS
 2 % for Lewandowsky and Farrell's
 3 % "Computational Modeling in Cognition: Principles ↩
     and Practice"
 4  global consts;
 5
 6 consts.seed = 2135; %for random generator
 7 consts.lSize=6;     %lineup size
 8 consts.nRep=1000;   %number of reps at each call
 9 consts.n=100;       %number of features in vectors
10 consts.nCond=9;     %number of conditions modeled
11 consts.fChoice=[7 8 9];   %forced-choice conditions
12 consts.paLineup=[4 5 6];  %paLineup conditions
13 consts.ptToCrit=[3 4 5 3 4 5]; %slots in parameters
14
15 %Data Exp 1 & 2 of Clare & Lewandowsky (2004),
16 %columns are: Suspect, Foil, and Reject
17 data = [.80, .13, .07;    %PP control
18         .57, .06, .36;    %PP holistic
19         .69, .12, .19;    %PP featural
20         .05, .72, .23;    %PA control
21         .20, .28, .52;    %PA holistic
22         .00, .48, .52;    %PA featural
```

(Continued)

(Continued)

```
23           .86, .14, .00;    %Exp 2 control
24           .81, .19, .00;    %Exp 2 holistic
25           .84, .16, .00];   %Exp 2 featural
26
27 %initialize parameters in order:
28 %  s
29 %  sim
30 %  crec -control
31 %  crec -holistic
32 %  crec -featural
33 disp ('Starting values of parameters')
34 startParms = [0.2942    0.3508    1.0455    2.0930    ↩
       1.8050]
35 consts.maxParms = [1. 1. inf inf inf];
36
37 [finalParms,fVal] = Wwrapper4fminBnd(startParms, ↩
       data);
38
39 %print final predictions
40 predictions = witness(finalParms)
```

The next few lines (17–25) initialize a matrix with the to-be-fitted data that were shown earlier in Table 7.1. You will note that all numbers in the table are also shown here, albeit in a slightly different arrangement (e.g., the columns represent response types rather than conditions) that simplifies the programming. Note also that the order of conditions, from top to bottom, is the same as their order, from left to right, in Figure 7.2. This is no coincidence because it means that the predictions returned by function `witness` share the layout of the data.

We next set the starting values for the parameters (line 34) and determine their maximum values (line 35) to ensure that they do not go out of bounds during estimation (e.g., sim must not exceed 1 because the similarity between two vectors cannot be greater than identity). Finally, `Wwrapper4fminBnd` is called to estimate the parameters (line 37). This part of the code is virtually identical to the example presented earlier (in Section 3.1.2) and does not require much comment. We therefore now turn to the remaining function, represented by the central box in Figure 7.1, which coordinates the parameter estimation.

7.1.3.4 Estimating the Parameters

Using MATLAB's standard search function. The function `Wwrapper4fmin` in Listing 7.4 should be quite familiar from the earlier chapters. Indeed, with the exception of specifying some options (in lines 4 and 5) and dealing with boundary conditions (line 9), this listing is nearly identical to Listing 3.2.

Listing 7.4 Parameter Estimation Function for WITNESS Simulation

```
 1 function [x, fVal] = Wwrapper4fmin(parms, data)
 2 global consts;
 3 
 4 defOpts = optimset ('fminsearch');
 5 options = optimset (defOpts, 'Display', 'iter', ↩
       'MaxFunEvals', 400)
 6 [x,fVal,dummy,output] = ↩
       fminsearch(@bof,parms,options,data)
 7 
 8     function rmsd=bof(parms, data)
 9         if (min(parms) < 0) || (min(consts.maxParms − ↩
               parms) < 0)
10             rmsd = realmax;
11         else
12             sd=(witness (parms)−data).^2;
13             rmsd=sqrt (sum(sum(sd))/numel(data));
14         end
15     end
16 end
```

The test for boundary conditions in line 9 is noteworthy: If any of the parameters are out of bounds, the function `bof` returns the maximum number that MATLAB can represent (realmax), thus signaling Simplex not to go anywhere near those values. Only if the current parameter values are within bounds are the predictions of WITNESS computed and compared to the data by computing the standard RMSD (Equation 2.2). We used RMSD as the discrepancy function for comparability with Clare and Lewandowsky (2004); because the data consist of counts (i.e., number of subjects who make a certain response), we could equally have used a χ^2 discrepancy function (which was employed by Clark, 2003).

Improving boundary checks. One limitation of the boundary check in Listing 7.4 is that it creates a "step" function, such that any legitimate parameter value, no matter how close to the boundary, is left unpenalized, whereas any out-of-bounds value, no matter how small the transgression, is given an equally large penalty. These problems can be avoided by using the `fminsearchbnd` function, which is not part of a standard MATLAB installation but can be readily downloaded from MATLAB Central.

Listing 7.5 shows an alternative parameter estimation function, called `Wwrapper4fminBnd`, which uses `fminsearchbnd` and passes the lower and upper bounds of the parameters as additional arguments (lines 4 and 5). Owing to the use of `fminsearchbnd`, the code has also become more compact because the boundary check did not have to be programmed explicitly.

Listing 7.5 Using `fminsearchbnd` for WITNESS Simulation

```
function [x, fVal] = Wwrapper4fminBnd(parms, data)
global consts;

[x,fVal,dummy,output] = ↩
    fminsearchbnd(@bof,parms,zeros(size(parms)),
           consts.maxParms)

    function rmsd=bof(parms)
        sd=(witness (parms)—data).^2;
        rmsd=sqrt (sum(sum(sd))/numel(data));
    end
end
```

There is one other noteworthy aspect of `Wwrapper4fminBnd`: The embedded function `bof` accesses the data not as a function argument (as in Listing 7.4) but by referring to a variable `data` that was passed to the surrounding function. We already discussed this valuable ability of embedded functions to automatically inherit variables from the surrounding function in Section 3.1.2; if this sounds cryptic, you may wish to reread that earlier section. For didactic reasons, we used the inheriting regime in one of our functions while using function arguments in the other.

7.1.4 WITNESS Simulation Results

Our modeling seeks to answer two questions: First, does the criterion explanation as embodied in WITNESS account for the data of Clare and Lewandowsky (2004) at a quantitative level? Second, can an alternative memory explanation be constructed within WITNESS to handle those data?

7.1.4.1 The Criterion Explanation

The simulation as just described was run three times, with a different set of starting values for the parameters on each run. The best-fitting estimates reported in Table 7.2 are based on the run that yielded the smallest RMSD (.0498), suggesting that the average deviation between model predictions and data was on the order of 5%.

Can we be certain that this solution reflects a global rather than a local minimum? There can be no complete assurance that the observed minimum is indeed global, but several facts raise our level of confidence in the solution. First, the different runs converged on very similar RMSDs—namely, .0498, .0511, and

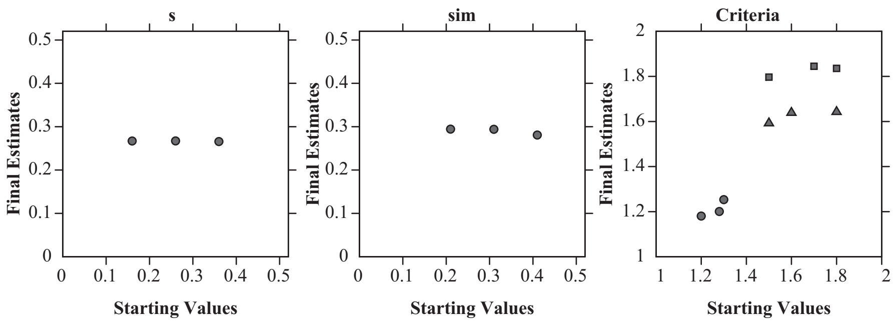

Figure 7.3 Final parameter estimates as a function of their starting values for three fits of the WITNESS model to the data of Clare and Lewandowsky (2004). From left to right, the panels show the values of s, sim, and the recognition criteria, respectively. In the rightmost panel, circles, squares, and triangles refer to $C_{rec}(C)$, $C_{rec}(H)$, and $C_{rec}(F)$, respectively.

.0509, respectively. Second, the final parameter estimates were remarkably similar, notwithstanding considerable variation in their starting values. This is illustrated in Figure 7.3, which shows the final parameter estimates as a function of their starting values. The figure shows that there is very little variability along the ordinate (final estimates) even though the values differ quite noticeably along the abscissa (starting values). The fact that the three simulation runs converged onto nearly indistinguishable best-fitting estimates bolsters our confidence that we have reached a global minimum.

How well does the model capture the data? Figures 7.4 and 7.5 show the data from Experiments 1 and 2, respectively, together with the model predictions.

It is clear from the figures that the model handles the main trends in the data and provides a good quantitative fit of both experiments. By varying the decision criterion, WITNESS captures the effects of verbalization on identification performance for both optional-choice and forced-choice lineups and for perpetrator-absent as well as perpetrator-present lineups. This result lends considerable support to the criterion explanation of verbal overshadowing. What remains to be seen is whether a memory-based alternative explanation may also handle the results.

7.1.4.2 An Alternative Memory Explanation

Clare and Lewandowsky (2004) also examined whether WITNESS might handle the results by assuming that memory is modified during verbalization. Specifically, to instantiate a memory explanation, Clare and Lewandowsky assumed that verbalization partially overwrites the perpetrator's image in memory. This was modeled by a reduction in the encoding parameter s because reducing s is

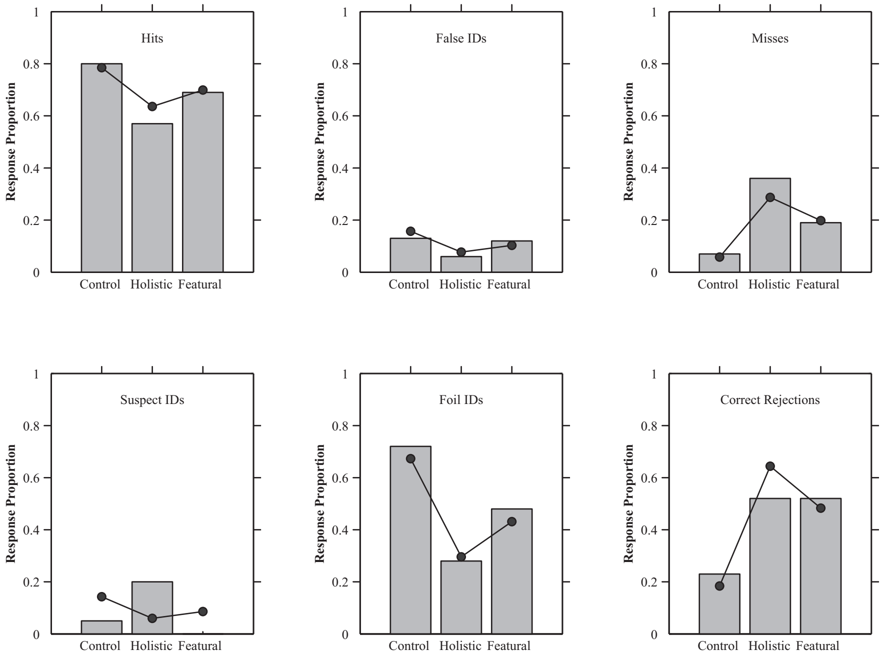

Figure 7.4 Data (bars) and predictions of the criterion explanation within WITNESS (points and lines) for Experiment 1 (optional-choice lineups) of Clare and Lewandowsky (2004). The top row of panels represents the perpetrator-present lineup and the bottom row the perpetrator-absent lineup. Data from Clare, J., & Lewandowsky, S. (2004). Verbalizing facial memory: Criterion effects in verbal overshadowing. *Journal of Experimental Psychology: Learning, Memory, & Cognition, 30*, 739–755. Published by the American Psychological Association; adapted with permission.

equivalent to unimpaired encoding followed by overwriting of the encoded features.

We instantiated the same idea in our simulations. To conserve space, we do not show the modified source code, although it is available at the book's supporting webpage (http://www.cogsciwa.com). In a nutshell, the criterion parameter (c_{rec}) was kept constant across all conditions, whereas the encoding parameter (s) differed between the control, holistic, and featural conditions. This version of WITNESS failed to handle the data, as shown in Figure 7.6. The figure shows predictions with the best-fitting parameter estimates reported by Clare and Lewandowsky (2004). We do not present the fit to Experiment 2 as it does not deviate appreciably from that of the criterion explanation.

Unlike the criterion explanation, the memory explanation cannot handle the effects of verbal overshadowing, presumably because it cannot simultaneously explain the improved performance for perpetrator-absent lineups and the

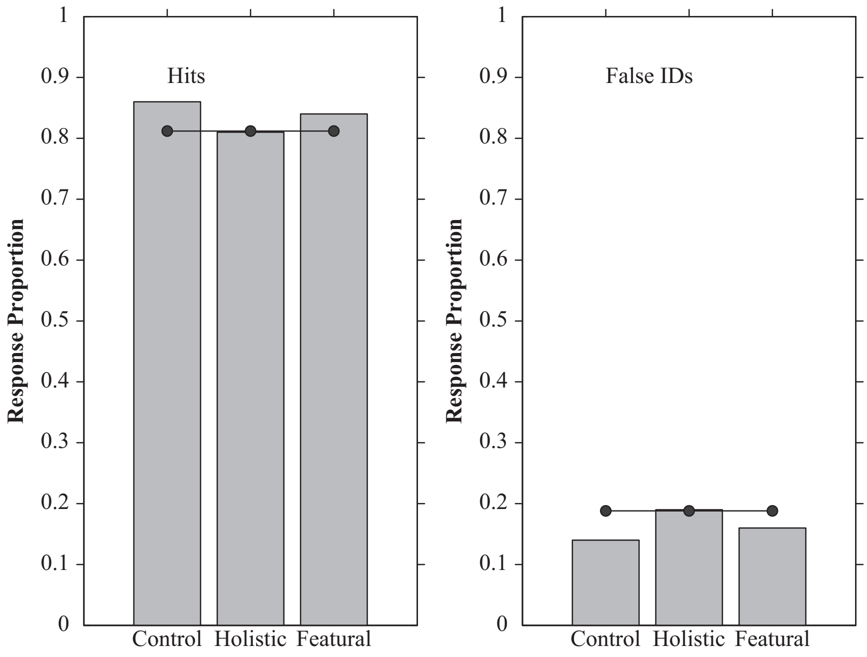

Figure 7.5 Data (bars) and predictions of the criterion explanation within WITNESS (points and lines) for Experiment 2 (forced-choice lineup) of Clare and Lewandowsky (2004). Data from Clare, J., & Lewandowsky, S. (2004). Verbalizing facial memory: Criterion effects in verbal overshadowing. *Journal of Experimental Psychology: Learning, Memory, & Cognition, 30*, 739–755. Published by the American Psychological Association; adapted with permission.

detriment to performance with perpetrator-present lineups. We invite you to modify the source code just presented in order to instantiate the memory explanation and to reproduce the results in the figures. If you encounter difficulties or obtain surprising results, you can consult the information at our webpage. Clare and Lewandowsky (2004) applied the model to all three of their experiments simultaneously, and they estimated parameters only based on perpetrator-present lineups; you will get different results if you fit two experiments only or if you fit all conditions.

7.1.4.3 Conclusions From the Modeling

What conclusions can we draw from the modeling involving WITNESS? First, the modeling shows that the verbal overshadowing effect arguably reflects a criterion adjustment rather than an impairment of memory after verbalization. When this

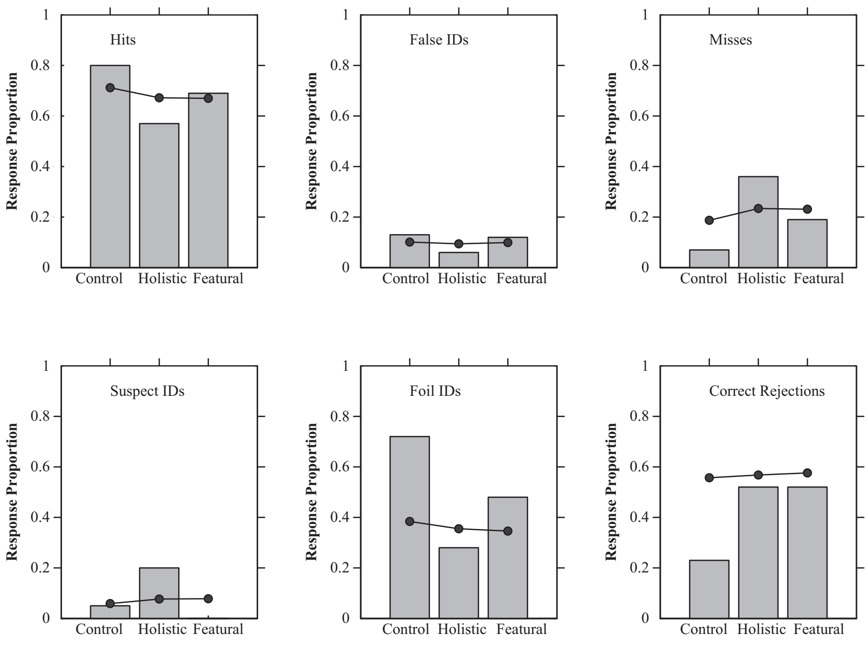

Figure 7.6 Data (bars) and predictions of the memory explanation within WITNESS (points and lines) for Experiment 1 (optional-choice lineups) of Clare and Lewandowsky (2004). The top row of panels represents the perpetrator-present lineup and the bottom row the perpetrator-absent lineup. Data from Clare, J., & Lewandowsky, S. (2004). Verbalizing facial memory: Criterion effects in verbal overshadowing. *Journal of Experimental Psychology: Learning, Memory, & Cognition, 30*, 739–755. Published by the American Psychological Association; adapted with permission.

criterion explanation is instantiated in WITNESS, the model can predict both the presence and absence of verbal overshadowing (or indeed a beneficial effect for perpetrator-absent lineups) depending on the type of decision that is expected of participants. Second, the modeling showed that the data are not readily modeled by an explanation based on alteration or overwriting of memory during verbalization. Although this second finding does not rule out a memory-based explanation because we did not explore all possible instantiations of that alternative, the fact that the criterion explanation handles the data quite parsimoniously makes it an attractive account of the phenomenon.

In the context of verbal overshadowing, Clark (2008) drew attention to the fact that the mere label for the effect—namely, "verbal *overshadowing*"—is not theory neutral but, simply by its name, "implies the memory impairment explanation that has been the dominant explanation . . . since the original results were reported" (pp. 809–810). Thus, merely giving an effect a name can create the *appearance* of an explanation: Far from being an advantage, this erroneous appearance may bias subsequent theorizing and may retard corrective action (cf. Hintzman, 1991). One of the uses of computational modeling is to provide *substantive* explanations that necessarily go beyond labeling of an effect. Thus, although we refer to the "criterion explanation" by name, this is a name not for a phenomenon but for a fully instantiated process explanation.

One limitation of the example just presented is that we only considered a single model (albeit in two different variants). In general, stronger inferences about cognitive processes are possible if several competing models are compared in their ability to handle the data. The next example illustrates this situation.

7.2 Exemplar Versus Boundary Models: Choosing Between Candidates

One central aspect of human cognition—and shared with such animals as primates and pigeons (e.g., Fagot, Kruschke, Depy, & Vauclair, 1998; Farrell et al., 2006; Shimp, Long, & Fremouw, 1996)—is the ability to categorize a virtually limitless range of objects in the world into a much smaller number of relevant categories (e.g., Ashby & O'Brien, 2005; Bruner, Goodnow, & Austin, 1956; Lewandowsky, Kalish, & Ngang, 2002; Murphy, 2002; Nosofsky, 1986; Palmeri, Wong, & Gauthier, 2004). Doing so leads to more compact and efficient representations of the world; we can ignore trivial differences between objects and deal with them on the basis of their category. Knowing whether something is a chair or a table is important for determining whether we sit at it or on it; knowing whether it is made of oak or pine is not. Unsurprisingly, a huge wealth of data has been generated about people's learning and use of categories, and this has driven—and been driven by—considerable theoretical development (see, e.g., Figure 1 of Palmeri et al., 2004). The area of categorization is specifically notable in having birthed a number of formal models, including GCM (e.g., Nosofsky, 1986), GRT (e.g., Ashby, 1992b), ALCOVE (Kruschke, 1992), RASHNL (Kruschke & Johansen, 1999), the EBRW model (Nosofsky & Palmeri, 1997), and COVIS (Ashby, Alfonso-Reese, Turken, & Waldron, 1998). You've already seen one of these models, the Generalized Context Model (GCM), and its approach to explaining category learning in Chapter 1.

In the following, we will examine three models of categorization and present code for maximum likelihood estimation of the parameters of those models, along with calculation of the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC) for the purposes of model comparison.

7.2.1 Models of Categorization

Research on categorization has considered a wide variety of category structures of varying dimensionality. Here, we will assume that category members vary along a single dimension. This will simplify the presentation of the models, but it will nonetheless allow us to make some very diagnostic inferences based on a unidimensional categorization experiment that we will discuss a little later.

7.2.1.1 Generalized Context Model: GCM

You've already been presented with the basics of GCM in Chapter 1. Because that was a while ago, and so that you can easily refer back to this material when reading about the application of GCM below, we'll briefly summarize the model again. GCM's main claim is that whenever we have an experience about an object[6] and its category, we store a localist, unitary representation of that object and its category: an exemplar. For the example we are looking at here, the experimental stimuli only vary along a single dimension, and so only a single feature is relevant.

When we come across an object in the world and wish to determine its category (e.g., edible vs. inedible), the GCM postulates that we match that object to all exemplars in memory and categorize the object according to the best match to existing exemplars. This matching process relies on a calculation of the distance between the object i and all the stored exemplars $j \in \mathfrak{J}$:

$$d_{ij} = |x_i - x_j|. \tag{7.2}$$

Note the simple form of this equation compared to the earlier Equation 1.3. We've been able to simplify this because here we only have a single stimulus dimension. This means we do not need to sum across several dimensions, and it also means that we can leave out the generalization to different types of difference metric: For a single dimension, $|a - b| = \sqrt{(a-b)^2}$. This distance is then mapped into a measure of similarity (i.e., match) between the new stimulus and each exemplar:

$$s_{ij} = exp(-c \cdot d_{ij}). \tag{7.3}$$

Remember that the c in Equation 7.3 scales the drop-off in similarity with increasing distance. When c is small, a stimulus will match a wide range of exemplars (i.e., a slow drop-off with distance), and when c is large, the stimulus will only match exemplars within a very narrow range (i.e., a fast drop-off with distance). Finally, the similarity values are added up separately for exemplars in each category and used to determine the categorization probability for the new stimulus:

$$P(R_i = A|i) = \frac{\left(\sum\limits_{j\in A} s_{ij}\right)}{\left(\sum\limits_{j\in A} s_{ij}\right) + \left(\sum\limits_{j\in B} s_{ij}\right)}. \tag{7.4}$$

7.2.1.2 General Recognition Theory: GRT

The fundamental assumption of the GCM is that all past experiences with category members are stored as exemplars in memory. This assumption permits the model to explain the relationship between categorization and recognition memory as discussed in Chapter 1. General recognition theory (GRT; Ashby, 1992a; Ashby & Gott, 1988), by contrast, takes a very different tack. The GRT assumes that what is represented in memory is an abstraction of the category structure rather than the exemplars themselves. Specifically, GRT assumes that the stimulus space is carved into partitions by decision boundaries. All that needs to be stored in memory in order to classify new exemplars is some specification of the placement of the boundaries. In multidimensional space, this can get quite complicated to conceptualize, as the boundaries can have any form, although they are usually assumed to be specified by linear or quadratic equations.[7] In the case of stimuli that vary along only a single dimension, things are very easy: A category boundary is a single point along the stimulus dimension.

Categorization errors are assumed to follow from trial-by-trial variability in the perception of the stimulus (Alfonso-Reese, 2006). This means that on one trial, a stimulus may appear to fall on one side of the boundary, whereas on another trial, it appears to fall on the other side. GRT assumes that this variability takes the form of a normal probability density function (PDF) centered on the true value of the stimulus and with standard deviation σ. An example is shown in the top panel of Figure 7.7: The circle shows the actual stimulus value, and around that is drawn the normal PDF.

Given the normal density around the stimulus and the placement of the boundaries, what is the probability of categorizing the stimulus as belonging to a particular category? This can be worked out using concepts we discussed in Chapter 4 with reference to probability functions. Recall that in the case of a PDF, the probability that an observation will fall within a specific range involves finding the area under the curve within that range. That is, we need to integrate the PDF between the minimum and maximum values defining that range. Take a look again at the top panel of Figure 7.7. The portion of the PDF shaded in gray shows the area under the curve to the left of the boundary; this area under the PDF is the probability that the perceived stimulus value falls below the boundary and therefore corresponds to the probability that the stimulus i will be categorized as an 'A,'

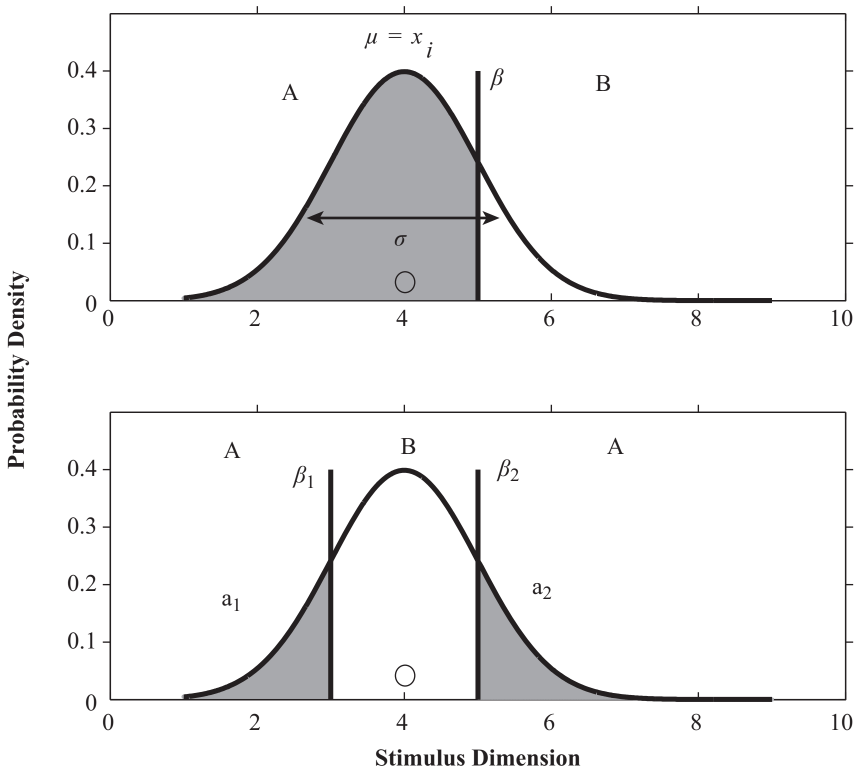

Figure 7.7 Depiction of one-dimensional categorization in GRT. Both panels show a normal PDF reflecting the variability in perception of the stimulus; the stimulus itself has a value of 4 and is shown as a circle. The vertical lines in each panel (labeled β) show the category boundaries. In the top panel, only a single boundary exists, and the probability of categorizing the stimulus as being in Category A is the area under the normal density to the left of the boundary. The bottom panel shows a more complicated example with two boundaries; in this case, the probability of categorizing the stimulus as an ‘A’ is the area under the curve to the left of the left-most boundary (β_1) plus the area to the right of the right-most boundary (β_2).

$P(R_i = A|i)$. In the top panel, we need to find the integral from the minimum possible stimulus value up to the boundary value (β). Let’s assume that the stimulus dimension is unbounded, so that the minimum possible value is $-\infty$. This means we need to calculate

$$P(R_i = A|i) = \int_{-\infty}^{\beta} \mathrm{N}(x_i, \sigma), \tag{7.5}$$

where N is the normal PDF with mean x_i and standard deviation σ. To calculate this integral, we can use the normal cumulative distribution function (CDF). As we discussed in Chapter 4, the CDF is the integral of a PDF and can therefore be used to integrate across segments of the PDF. Using the integral of the normal CDF, Φ, we can rewrite Equation 7.5 as

$$P(R_i = A|i) = \Phi\left(\frac{\beta - x_i}{\sigma}\right), \tag{7.6}$$

where the integration is assumed to be taken from $-\infty$. The argument passed to the normal CDF, $(\beta - x_i)/\sigma$, expresses the boundary as a z score of the normal density around the stimulus because the normal CDF function assumes a mean of 0 and a standard deviation of 1.

We can also use this method to obtain predicted probabilities from GRT for more complicated examples. The bottom panel of Figure 7.7 shows a case where there are two boundaries, β_1 and β_2. Stimuli below β_1 and above β_2 belong to Category A, whereas stimuli between the two boundaries belong to Category B; such a category structure might arise when determining whether some milk is safe to feed to an infant given its temperature. The predicted probability of categorizing a stimulus i as being an 'A' is then the probability that either the perception of the stimulus falls below β_1 or that it falls above β_2. These two probabilities are mutually exclusive (unless it is quantum event, the stimulus cannot simultaneously both fall below β_1 and above β_2), so following the rules of probability in Chapter 4, the probability of either event happening can be obtained by adding up the individual probabilities; that is, we sum the gray areas in Figure 7.7. The first area is obtained in a similar manner to the top panel, by integrating from $-\infty$ to β_1:

$$P(R_i = a_1|i) = \Phi\left(\frac{\beta_1 - x_i}{\sigma}\right). \tag{7.7}$$

The second region, to the right of β_2, requires only a little more thinking. The CDF gives the integral from $-\infty$ up to some value, so we can use it to obtain the integral up to β_2. To work out the area above β_2, we can calculate the integral up to β_2 and subtract it from 1: Remember that probabilities must add up to 1, and if the perceived stimulus doesn't fall to the left of β_2, it must fall to the right (we are assuming that the perceived stimulus cannot fall directly on the boundary). Written as an equation, it looks like this:

$$P(R_i = a_2|i) = 1 - \Phi\left(\frac{\beta_2 - x_i}{\sigma}\right). \tag{7.8}$$

The probability of making an 'A' response is then:

$$P(R_i = A|i) = P(R_i = a_1|i) + P(R_i = a_2|i).$$

Although we can determine the optimal placement of boundaries, we cannot a priori specify the boundaries a participant is actually using. Accordingly, boundary placement will be usually specified by free parameters. In the case of unidimensional stimuli, each boundary will be captured by a single free parameter, and there will be an additional free parameter for σ, the variability in perception.

7.2.1.3 Deterministic Exemplar Model: DEM

Much research has been directed at discriminating between GCM and GRT as competing explanations for human (and animal) categorization (e.g., Farrell et al., 2006; Maddox & Ashby, 1998; McKinley & Nosofsky, 1995; Nosofsky, 1998; Rouder & Ratcliff, 2004). Ashby and Maddox (1993) and Maddox and Ashby (1993) noted that one difficulty with comparing GCM and GRT directly is that such a comparison confounds the representations involved in categorizing a stimulus and the processes required to turn the resulting information into an overt categorization response. That is, not only do GCM and GRT obviously differ in the way category information is represented, but they also differ in the way in which responses are generated. In GCM, responses are probabilistic (by virtue of the use of the Luce choice rule, Equation 7.4), whereas GRT's responses are deterministic: If a stimulus is perceived to fall to the left of the boundary in the top panel of Figure 7.7, it is always categorized as an 'A.'

To partially deconfound these factors, Ashby and Maddox (1993) presented a deterministic exemplar model (DEM). This model is identical to the standard GCM model, with the exception that the response rule is deterministic. DEM replaces Equation 7.4 with a modified version of the Luce choice rule:

$$P(R_i = A|i) = \frac{\left(\sum\limits_{j \in A} s_{ij}\right)^{\gamma}}{\left(\sum\limits_{j \in A} s_{ij}\right)^{\gamma} + \left(\sum\limits_{j \in B} s_{ij}\right)^{\gamma}}. \tag{7.9}$$

The modification is that the summed similarities are raised to the power of a free parameter γ. This parameter controls the extent to which responding is deterministic. If $\gamma = 1$, the model is identical to GCM. As γ gets closer to 0, responding becomes more and more random, to the point where γ=0 and Equation 7.9 works out as $1/(1+1)$: Responding is at chance and isn't sensitive to the actual stimulus presented. As γ increases above 1, responding gets more and more deterministic. This is because γ acts nonlinearly on the summed similarities, and if $(\sum\limits_{j \in A} s_{ij})^{\gamma}$ is greater than $(\sum\limits_{j \in B} s_{ij})^{\gamma}$, then increasing γ will increase $(\sum\limits_{j \in A} s_{ij})^{\gamma}$ more than it increases $(\sum_{j \in B} s_{ij})^{\gamma}$. For a very large value of γ, the model will respond deterministically: If $(\sum_{j \in A} s_{ij}) >> (\sum_{j \in B} s_{ij})$, the model will always produce an 'A' response.

7.2.2 A Probabilistic Feedback Experiment

How, then, can the boundary and exemplar models be teased apart? Can they be unambiguously differentiated? Rouder and Ratcliff (2004) approached this problem with an ingenious paradigm in which stimuli varied along a single dimension. Rouder and Ratcliff (2004) presented their participants with a 640 $\times$ 480 grid of pixels, with each pixel being colored white, black, or gray. Rouder and Ratcliff varied the proportion of nongray (black or white) pixels that were white and asked participants to categorize each grid as being light or dark. Following Rouder and Ratcliff, we call this single-stimulus dimension *luminance* from here on.

The critical feature for telling apart boundary and exemplar models was that category membership was probabilistic: The same stimulus x_i was sometimes designated as being in Category A by feedback during training and sometimes as being in Category B. The top panel of Figure 7.8 shows the category structure for Rouder and Ratcliff's Experiment 1. The black dots on the solid line refer to the locations, in luminance space, of the eight training stimuli that we number 1 through 8 (from left to right). Stimuli at either end of the stimulus space (i.e., 1, 2, 7, and 8) were presented as members of Category A on 60% of the trials. Stimuli in the middle region of the stimulus space either belonged to Category A 100% of the time (stimuli 3 and 4) or were always Category B members (i.e., never Category A; stimuli 5 and 6).

The bottom two panels show representative predictions from boundary and exemplar theories and show how the experiment can discriminate between the different theories. The middle panel shows predictions from GRT, under the assumption that people place the boundaries more or less optimally. Although participants cannot reach perfect accuracy in this task given its probabilistic nature, they can nonetheless maximize performance by inserting a boundary at each point at which the probability of belonging to Category A in the top panel shifts from above .5 to below .5. That is, the best strategy would be to place one boundary between stimuli 4 and 5 and another one between stimuli 6 and 7. Although there is a shift in probabilities between stimuli 2 and 3, placing a boundary here would be suboptimal because it would lead to a tendency for participants to classify stimuli 1 and 2 as being in Category B, whereas they are more likely to belong to Category A.

Under these assumptions about boundary placement, the middle panel shows that GRT predicts a function that dips in the middle and is monotonically

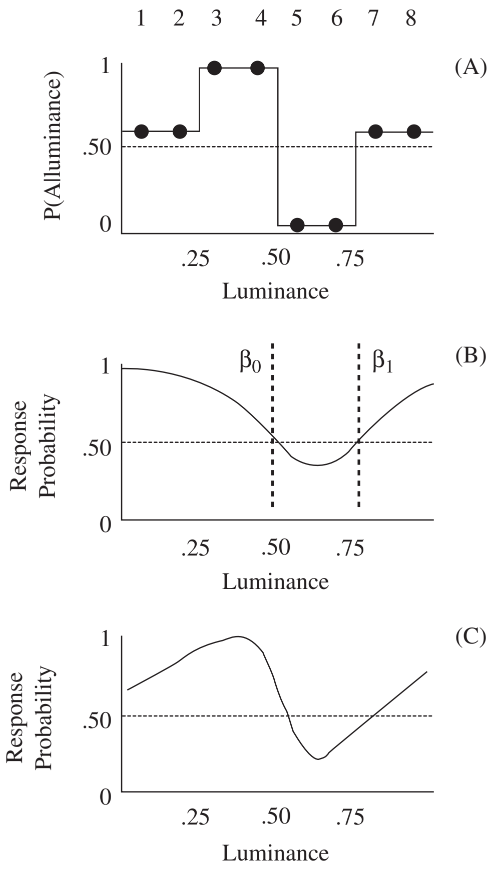

Figure 7.8 Experimental structure and model predictions for Rouder and Ratcliff's (2004) probabilistic categorization experiment. The top panel shows the proportion of times each stimulus value was designated as belonging to Category A by experimental feedback; the integers above the plot number the stimuli from 1 to 8. The middle panel shows representative predictions from GRT under the assumption that a boundary is placed wherever the membership probability crosses 0.5. The bottom panel shows representative exemplar model predictions, from DEM. Figure reprinted from Rouder, J. N., & Ratcliff, R. (2004). Comparing categorization models. *Journal of Experimental Psychology: General*, *133*, 63–82. Published by the American Psychological Association; reprinted with permission.

increasing as one moves away from the dip in either direction. In contrast, the bottom panel shows that an exemplar model (DEM) will predict a more complicated function. Although the same dip occurs for stimuli 5 and 6, there is a second dip at the bottom end of the stimulus space.

Why do the two models make different predictions? Exemplar theory predicts that response probabilities should track the feedback probabilities. As we move down the stimulus dimension in the lower half of the space (i.e., from stimulus 4 down to stimulus 1), the proportion of exemplars in memory that belong to Category A drops from 1.0 to 0.6. This means that the summed similarity for Category A will also drop, and since the summed similarity feeds directly into the predicted responses via Equation 7.4 or Equation 7.9, the response proportions are also expected to drop. The extent of this drop will depend on the parameter settings and on the particular model: GCM will show a strong tendency to track the feedback probabilities, while DEM, with its ability to respond deterministically, may show responding that is more deterministic (Farrell et al., 2006). GRT, by contrast, predicts that as we move away from a stimulus boundary the more likely a stimulus is to be classified as corresponding to that category. Although the membership probability decreases as we move to the left of the top panel of Figure 7.8, all GRT is concerned with is the placement of the boundaries; as we move to the left, we move away from the boundary, and the predicted probability of classifying the stimuli as 'A's necessarily increases in a monotonic fashion. A defining characteristic of GRT is that it cannot predict non-monotonicity in response proportions as the absolute distance of a stimulus from the boundary increases.

Figure 7.9 shows the results from the last three sessions for the six participants from Rouder and Ratcliff's (2004) experiment, along with the feedback probabilities reproduced from Figure 7.8 (in gray). The participants in the left and middle columns of Figure 7.9 (SB, SEH, BG, and NC) qualitatively match the predictions of GRT, particularly in the monotonic shape of their response probability functions in the left-hand size of the stimulus space. Participant VB (top right panel) behaves more or less as predicted by GRT, but with a small downturn for the smallest luminances as predicted by exemplar theory. Finally, participant LT (bottom right panel) behaves as predicted by exemplar theory, with a very clear downturn in 'A' responses.

Following Rouder and Ratcliff (2004), we next apply exemplar and decision-bound models to the data from the six participants shown in Figure 7.9. We will use maximum likelihood estimation (MLE) to fit the models to the data and obtain standard errors on the parameter estimates for each participant. Finally, we will use AIC and BIC to compare the models on their account of the data.

7.2.3 MATLAB Code for ML Parameter Estimation for GCM, DEM, and GRT

Let's start with the code for the log-likelihood functions from GCM, DEM, and GRT. We will then move on to look at how we use this code to compare models.

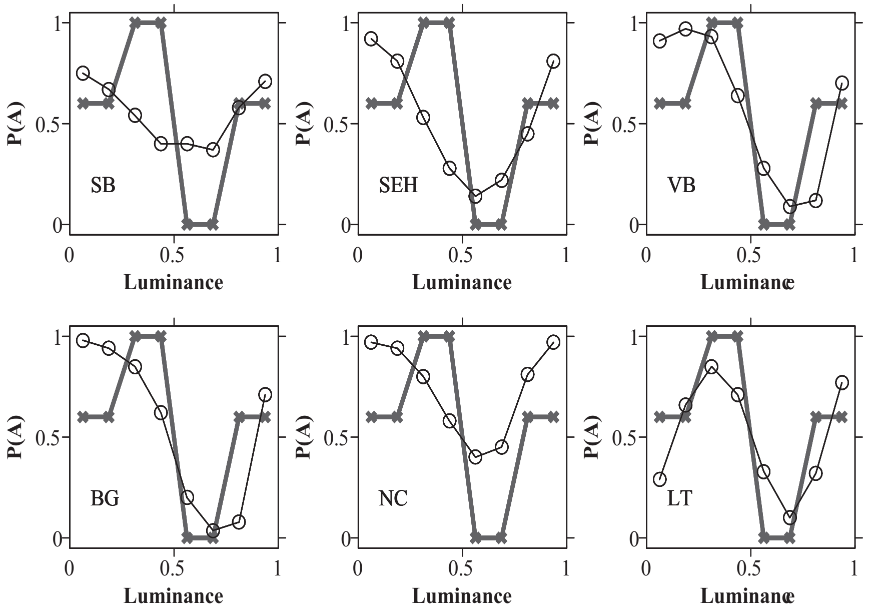

Figure 7.9 Proportion of trials on which stimuli were categorized as belonging to Category A, for six participants (separate panels). Feedback probabilities from the training session are shown in gray. Figure adapted from Rouder, J. N., & Ratcliff, R. (2004). Comparing categorization models. *Journal of Experimental Psychology: General*, *133*, 63–82. Published by the American Psychological Association; reprinted with permission.

7.2.3.1 GCM and DEM

Listing 7.6 provides the code for calculation of the negative log-likelihood for GCM and DEM. Recall that GCM is simply the DEM model with γ set to 1. Accordingly, we can use the DEM code to run both GCM and DEM, as long as we remember to set γ equal to 1 when running GCM. The first several lines are comments to tell us what the input arguments are. The first input argument is the parameter vector `theta`, which contains c as the first element (line 9) and γ as the second element (line 10). The second argument `x` is a vector containing the luminance values for the eight possible stimuli. The third argument `feedback` specifies how many times each stimulus was designated to be an 'A.' Specifically, `feedback` has two columns, one giving the number of times A was given as feedback for each stimulus and a second column giving the number of times B was designated as feedback. The fourth argument `data` contains the frequencies of A responses from the participant for each stimulus, and the vector `N` contains the total number of trials per stimulus in a vector.

Listing 7.6 Negative Log-Likelihood Function for GCM and DEM

```
 1 function [lnL, predP] = DEMlnL(theta, x, feedback, data, N)
 2 % gives the negative log-likelihood
 3 % for response frequencies from a single participant (data)
 4 % given the DEM parameters (in vector theta), the stimulus values in x,
 5 % the number of times each stimulus was an `A' in feedback
 6 % the number of times `A' was selected as a response (data)
 7 % and the total number of test trials for each stimulus (N)
 8
 9 c = theta(1);
10 gamma = theta(2);
11
12 for i=1:length(x)
13     s = exp(-c.*abs(x(i)-x));
14     sumA = sum(s.*feedback(1,:));
15     sumB = sum(s.*feedback(2,:));
16     predP(i) = (sumA^gamma)/(sumA^gamma + sumB^gamma);
17 end
18
19 lnL = -sum(data.*log(predP) + (N-data).*log(1-predP));
```

The GCM/DEM calculations are contained in lines 12 to 17. The loop goes through each of the stimuli, i, and for each stimulus calculates the predicted proportion of ‘A’ responses, `predP(i)`. Line 13 instantiates Equation 7.3, using the simple absolute distance measure in Equation 7.2. Line 13 is in a vectorized form: It calculates the similarity between each x_j and x_i in one go and returns a vector of similarities as a result. Lines 14 and 15 calculate summed similarity values. Rather than considering each exemplar individually, lines 14 and 15 take advantage of the design of the experiment in which there are only eight possible stimulus values. Since identical stimulus values will have identical similarity to x_i, we can calculate the summed similarity between x_i and all the exemplars with a particular value x_j by simply multiplying s_j by the number of exemplars with that particular value of x_j. Again, we don’t refer explicitly to s_j in the code because the code is vectorized and processes all js at once. Line 16 feeds the summed similarity values into the Luce choice rule with γ-scaling to give a predicted proportion correct (Equation 7.9).

The final line in Listing 7.6 calculates the negative log-likelihood based on the predicted proportions and observed frequencies. Do you recognize the form of line 19? If not, turn to Section 5.4. Otherwise, right! It's the binomial log-likelihood function we discussed in Chapter 4 and that we used to obtain probabilities from SIMPLE in Chapter 5. The function takes the observed frequencies, the total number of trials, and the predicted proportions to calculate a log-likelihood value for each stimulus (in vectorized form). We then sum the $\ln L$ values and take the negative so we can minimize the negative log-likelihood.

One issue that deserves mention is that we are using the simplified version of the binomial log-likelihood function and have omitted some of the terms from the full function (see Section 4.4). This has implications for calculation of AIC and BIC, as discussed in Chapter 5. This won't be a problem here, as we will use the same binomial data model for all three categorization models and will therefore be subtracting the same constant from AIC and BIC values for all three models.

Finally, to avoid confusion, it should be noted that Rouder and Ratcliff (2004) fit DEM to their data but called it GCM. This is because a number of authors treat DEM not as a model separate from GCM but as a version of GCM with response scaling. The differentiation between GCM and DEM is maintained here because of the possible importance of the decision process in the probabilistic nature of this experiment; in many circumstances, the nature of the response process will be a side issue separate from the main theoretical issue of concern.

7.2.3.2 GRT

Comprehensive code for the GRT is available in a MATLAB toolbox presented in Alfonso-Reese (2006). We will look at simpler code here that is aimed specifically at the experiment of Rouder and Ratcliff (2004). The MATLAB code for GRT is given in Listing 7.7 and has a similar structure to the MATLAB code for GCM. The first argument is a vector containing values for GRT's free parameters, which are extracted in the first bit of the code. The code opens assignment of the three elements of `theta` to `boundary1`, `boundary2` and `sd`. The function assumes that the first element in `theta` is the lower bound β_1 and that the second element is the upper bound β_2 and includes an `if` statement that performs a sanity check to ensure that the upper boundary is indeed greater than the lower boundary; if not, a large log-likelihood is returned. Lines 19 and 20 implement Equations 7.7 and 7.8. These calculations make use of the embedded function `normalCDF`, which makes use of the error function erf in MATLAB. The error function computes an integral that is related to the normal CDF by

$$\Phi(x) = \frac{1}{2}\text{erf}\left(\frac{x}{\sqrt{2}}\right). \tag{7.10}$$

This relationship is used to obtain the normal CDF in the embedded function `normalCDF` and was also used in Chapter 5 for the ex-Gaussian model, which also contains the normal CDF.[8] Line 21 adds the two areas `a1` and `a2` together to obtain predicted proportions, and line 23 implements the same binomial data function as was used in the GCM code. Notice that lines 19 to 21 are written in a vectorized format: At each step, the result is a vector whose elements correspond to specific stimulus values.

Listing 7.7 Negative Log-Likelihood Function for GRT

```
 1 function [lnL, predP] = GRTlnL(theta, x, data, N)
 2 % gives the negative log-likelihood
 3 % for response frequencies from a single participant ↩
       (data)
 4 % given the GRT parameters (in vector theta),
 5 % the stimulus values in x,
 6 % the number of times `A' was selected as a response ↩
       (data)
 7 % and the total number of test trials for each ↩
       stimulus (N)
 8
 9 bound1 = theta(1);
10 bound2 = theta(2);
11
12 if bound1 >= bound2
13     lnL = realmax;
14     predP = repmat(NaN, size(N));
15 end
16
17 sd = theta(3);
18
19 a1 = normalCDF((bound1-x)./sd);
20 a2 = 1 - normalCDF((bound2-x)./sd);
21 predP = a1 + a2;
22
23 lnL = -sum(data.*log(predP) + (N-data).*log(1-predP));
24
25 %% normalCDF
26 function p = normalCDF(x)
27
28 p = 0.5.*erf(x./sqrt(2));
```

7.2.4 Fitting the Models to Data

The code in Listings 7.6 and 7.7 takes a parameter vector, participant data, and other details about the experiment and returns the $-\ln L$ for the parameters given the data and the predicted proportion correct for each stimulus value given the parameters. This is sufficient to carry out parameter estimation and model selection. Listing 7.8 provides a script to fit each of the models and provides statistics for model comparison and further interpretation.

7.2.4.1 Setting Up

The first thing the script does is to assign the proportion of 'A' responses from the six participants to the variable `data`. The data are structured so that the rows correspond to participants (in the same order as in Figure 7.9), and each column corresponds to a stimulus i.

Listing 7.8 Code for MLE and Model Selection for the Categorization Models

```
% script catModels
clear all
close all

dataP = [0.75,0.67,0.54,0.4,0.4,0.37,0.58,0.71;
    0.92,0.81,0.53,0.28,0.14,0.22,0.45,0.81;
    0.91,0.97,0.93,0.64,0.28,0.09,0.12,0.7;
    0.98,0.94,0.85,0.62,0.2,0.037,0.078,0.71;
    0.97,0.94,0.8,0.58,0.4,0.45,0.81,0.97;
    0.29,0.66,0.85,0.71,0.33,0.1,0.32,0.77];

% number sessions x 10 blocks x 96 trials /(n stimuli)
Ntrain = ((5*10*96)/8);
pfeedback = [.6 .6 1 1 0 0 .6 .6];
Afeedback = pfeedback .* Ntrain;
feedback = [Afeedback; Ntrain-Afeedback];

Ntest = ((3*10*96)/8);
N = repmat(Ntest,1,8);

dataF = ceil(Ntest.*(dataP));

stimval = linspace(.0625, .9375, 8);

%% Maximum likelihood estimation
for modelToFit = {'GCM', 'GRT', 'DEM'};

    for ppt=1:6
        switch char(modelToFit)
            case 'GCM'
                f=@(pars) DEMlnL([pars 1], stimval, ↩
                    feedback, dataF(ppt,:), N);
                [theta(ppt,:),lnL(ppt),exitflag(ppt)]↩
```

```
                        =fminbnd(f, 0, 100);
            case 'GRT'
                f=@(pars) GRTlnL(pars, stimval, ↩
                    dataF(ppt,:), N);
                [theta(ppt,:),lnL(ppt),exitflag(ppt)]↩
                        =fminsearchbnd(f,[.3 .7 .1], ↩
                            [-1 -1 eps], [2 2 10]);
            case 'DEM'
                f=@(pars) DEMlnL(pars, stimval, ↩
                    feedback, dataF(ppt,:), N);
                [theta(ppt,:),lnL(ppt),exitflag(ppt)]↩
                        =fminsearchbnd(f,[5 1], [0 ↩
                            0], [Inf Inf]);
            otherwise
                error('Unknown model');
        end

        [junk, predP(ppt,:)] = f(theta(ppt,:));

        hess = hessian(f,theta(ppt,:),10^-3);
        cov = inv(hess);
        thetaSE(ppt,:) = sqrt(diag(cov));
    end

    figure
    pptLab = {'SB','SEH','VB','BG','NV','LT'};

    for ppt=1:6
        subplot(2,3,ppt);
        plot(stimval, dataP(ppt,:), '-+');
        hold all
        plot(stimval, predP(ppt,:), '-.*');
        ylim([0 1]);
        xlabel('Luminance');
        ylabel('P(A)');
        title(char(pptLab{ppt}));
    end
    set(gcf, 'Name', char(modelToFit));

    t.theta = theta;
    t.thetaSE = thetaSE;
    t.nlnL = lnL;
    eval([char(modelToFit) '=t;']);
    clear theta thetaSE
end


for ppt=1:6
```

(Continued)

(Continued)

```
77     [AIC(ppt,:), BIC(ppt,:), AICd(ppt,:), ↩
           BICd(ppt,:), AICw(ppt,:), BICw(ppt,:)] = ...
78         infoCriteria([GCM.nlnL(ppt) GRT.nlnL(ppt) ↩
               DEM.nlnL(ppt)], [1 3 2], ↩
               repmat(Ntest*8,1,3));
79 end
```

Line 13 works out an approximate number of training trials (i.e., number of exemplars per stimulus) based on details provided in Rouder and Ratcliff (2004): There were between four and six (i.e., approximately five) learning sessions per participant, each session containing 10 blocks of 96 trials that were shared between the eight stimuli. Line 14 assigns the category assignment probabilities in the top panel of Figure 7.8 to the variable `pfeedback`, and these are then multiplied with the number of trials stored in `Ntrain` to work out the number of trials on which the feedback indicated each stimulus belonged to Category A; this is assigned to the variable `Afeedback`. The variable `feedback` is then constructed in accordance with Listing 7.6, with one column for frequency of 'A' feedback and a second column for the frequency of 'B' feedback (these two values adding up to `Ntrain` for each luminance value).

Line 18 uses a similar procedure to line 13 to work out the number of test trials per stimulus. Rouder and Ratcliff (2004) analyzed the data from the last three sessions for each participant, by which time performance had stabilized. As for line 13, this is combined with the number of blocks per session (10) and the number of trials per block (96), along with the number of stimulus values (8) to determine the number of test trials per stimulus value per participant. It is important that this is exact, as these numbers will feed into the binomial log-likelihood function, which is sensitive to the overall number of trials (1,000 trials are more informative than only 10 trials and will sharpen the log-likelihood surface). These are then multiplied by the proportions stored in `dataP` to calculate `dataF`, the number of trials on which each participant responded 'A' for each stimulus value. The resulting value is then replicated eight times to make a vector `N` holding `Ntest` for each stimulus value.

The final piece of setting up is in line 23, which assigns the eight stimulus values (linearly spaced in the range .0625–.9375, with increments of .125) to the variable `stimval`. In line with Rouder and Ratcliff (2004), we assume that the physical luminance values (the proportion of nongray pixels that are white) map directly into psychological luminance values. However, it is possible that there may be some nonlinear relationship between the actual and perceived stimulus. Methods such as multidimensional scaling (Kruskal & Wish, 1978) are available to extract the underlying psychological dimension based on similarity ratings and confusability data. Rouder and Ratcliff (2004) compared a number of functions mapping physical luminance to perceived luminance and found all gave comparable results, including a function assuming a linear relationship between actual and perceived luminance. In providing the physical luminance values as perceived luminance, we are assuming a deterministic (i.e., noise-free) linear relationship between physical and perceived luminance.

7.2.4.2 Fitting the Models

The loop beginning at line 26 and finishing at line 73 does the hard graft of fitting each model to the data from each participant. We start off by setting up a loop across the models; each time we pass through the loop, we will be fitting a different model whose name will be contained in the variable `modelToFit` as a string. Within that loop is another loop beginning at line 28, which loops across participants, as we are fitting each participant's data individually, indexed by the loop variable `ppt`. The next line looks at `modelToFit` to work out which model's code to use in the fitting. This uses a switch statement, which matches `char(modelToFit)` to a number of possible cases ('GCM','GRT','DEM'), and `otherwise` reverts to a default catch statement, which here returns an error message. The switch statement uses `char(modelToFit)` rather than `modelToFit` directly because the set of values for `modelToFit` is provided in a cell array (being enclosed in curly braces), so the current value of `modelToFit` must be turned into a string using the function `char` before it can be matched to the string in each `case` statement.

Within each possible case, there are two lines. The first line (e.g., line 31) constructs an *anonymous function* (see the MATLAB documentation for further details). Anonymous functions are functions that are adaptively created on the fly and do not require their own function files in MATLAB. We are using anonymous functions here because, as we will see, we end up calling the same function several times. By using the anonymous function, we will simplify our code and make it easier to read. The main purpose of line 31 is to create a new on-the-fly function called `f` that takes only a single argument, a parameter vector called `pars`. Inside this function, all that happens is that we call `DEMlnL` and pass it `pars`, along with the other information needed by `DEMlnL`. Remember that GCM is DEM with γ fixed at 1; the first argument to DEM is therefore a vector joining together the free parameter c and the fixed value of 1 for γ. Going from left to right, we next have `stimval`, the luminance of each stimulus; `[feedback; Ntrain−feedback]`, a matrix containing one row for the number of times 'A' was given as feedback and a second row containing the number of times 'B' was given as feedback; the frequency of 'A' responses for the participant of current interest; and `N`, a vector specifying how many trials in total were tested for each stimulus. No feedback information is provided to the anonymous function for GRT because GRT does not require this argument. All this information from stimval onwards doesn't change within the participant loop; by using the anonymous function, we can specify it the one time and ignore it for the rest of the code, focusing instead on the parameter vector, which will need to change during the parameter estimation.

The second line within each case passes the anonymous function f to a minimization routine to find the ML parameter estimates (remember, we are minimizing the negative log-likelihood). GCM has only a single parameter to be estimated (c); accordingly, we use the built-in MATLAB function fminbnd, which performs unidimensional function minimization. This function does not need a starting point and only requires the minimum and maximum possible values for the parameter; we've set these to 0 and 100, respectively. For GRT and DEM, the models incorporating at least two free parameters, we use the fminsearchbnd function that was introduced in Section 7.1. This function requires a starting vector and vectors of lower and upper bounds on the parameters. These can be inspected in lines 37 and 41.

The result of what we've discussed so far is that for a given model (indexed by modelToFit), the code loops across participants, and for each participant, the anonymous function is reconstructed and passed to code that minimizes the appropriate negative log-likelihood function (contained in f) using a minimization routine. When each minimization attempt is finished, the ML parameter estimates, the minimized $-\ln L$, and the exit flag (giving information about whether or not the minimization was successful) are respectively placed in variables theta, lnL, and exitflag.

Having found the ML parameter estimates, line 46 runs the model a final time using the MLEs to obtain the predictions of the model under the MLEs; these predicted proportions of 'A' responses are stored in the matrix predP.

7.2.4.3 Obtaining Standard Errors on Parameter Estimates

After ML estimation has been performed, the next section of code (lines 48–50) finds the standard errors on the parameter estimates. Because we rewrote each log-likelihood function into the anonymous function f, we can use the same code for all three models. Line 48 passes the anonymous function and MLEs to the hessian function provided in Chapter 5, along with the δ parameter required by that function. The following two lines convert the resulting Hessian matrix hess into standard errors by taking the matrix inverse of the Hessian matrix to obtain a covariance matrix and then taking the square root of the values along the diagonal of this matrix to obtain the standard error on the ML parameter estimates.

The ML parameter estimates and their estimated standard errors are given in Tables 7.4, 7.5, and 7.6 for GCM, GRT, and DEM, respectively. We will refer back to these after discussing the model comparison results.

Table 7.4 ML Parameter Estimates and Associated Standard Errors for the GCM Fits to the Data in Figure 7.9

Participant	c	$SE(c)$
SB	2.29	0.36
SEH	5.29	0.32
VB	9.32	0.38
BG	9.43	0.38
NV	5.91	0.36
LT	10.28	0.41

Table 7.5 ML Parameter Estimates and Associated Standard Errors for the GRT Fits to the Data in Figure 7.9

Participant	β_1	$SE(\beta_1)$	β_2	$SE(\beta_2)$	σ	$SE(\sigma)$
SB	0.29	0.01	0.80	0.01	0.29	0.01
SEH	0.33	0.01	0.82	0.01	0.17	0.00
VB	0.92	0.01	0.46	0.01	0.16	0.00
BG	0.45	0.01	0.92	0.01	0.13	0.00
NV	0.43	0.01	0.69	0.01	0.15	0.00
LT	0.12	0.07	1.26	0.17	0.78	0.17

Table 7.6 ML Parameter Estimates and Associated Standard Errors for the DEM Fits to the Data in Figure 7.9

Participant	c	$SE(c)$	γ	$SE(\gamma)$
SB	2.44	0.90	0.95	0.26
SEH	4.89	0.43	1.13	0.11
VB	4.39	0.22	3.38	0.18
BG	4.65	0.21	3.24	0.16
NV	0.86	0.07	5.81	0.23
LT	13.51	1.19	0.69	0.07

7.2.4.4 Model Predictions

Lines 53 to 66 plot the predictions of the model currently specified by `modelToFit` given the ML parameter estimates. These plots are shown in Figures 7.10 to 7.12. GCM, whose predictions are shown in Figure 7.10, gives a visually poor fit to the data in most cases. GCM qualitatively misses the pattern in the data for participants SB, SEH, BG, and NV and apparently quantitatively

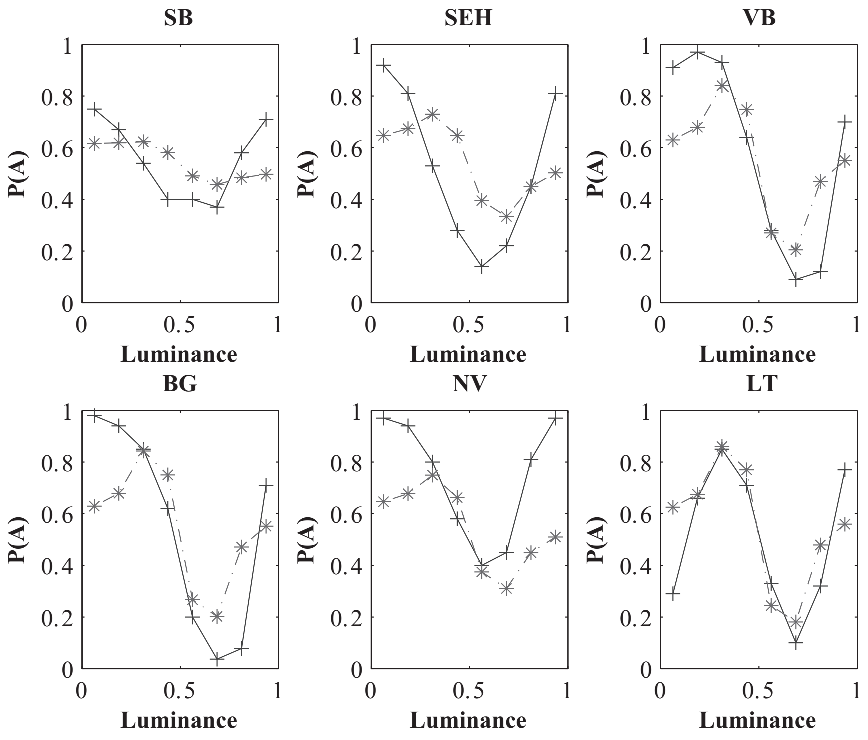

Figure 7.10 Proportion of 'A' responses predicted by GCM under ML parameter estimates. The data are plotted as crosses connected by solid lines, and the model predictions are plotted as asterisks connected by dashed lines.

misses the data of VB. The one participant for whom GCM appears to give a reasonable fit is participant LT in the bottom-right panel, who showed a very clear drop in 'A' responses with lower luminance values (stimuli 1 and 2). The predictions of GRT, shown in Figure 7.11, are much more in line with the data. The one exception is participant LT, for whom GRT predicts a nearly flat function that does little to capture the large changes in responses across luminance values for that participant. Finally, the predictions of DEM are shown in Figure 7.12. These predictions are similar to those of GCM (Figure 7.10), with DEM giving visually better fits in some cases (VB and BG). DEM generally appears to be inferior to GRT, with two exceptions. For participant VB, GRT and DEM appear to give equally good fits, and for participant LT, GRT is clearly inferior to DEM.

7.2.4.5 Model Comparison

Let's use the model selection methods presented in Chapter 5 to compare the three models. Although GRT looks like it gives a better fit to the data in some cases, this

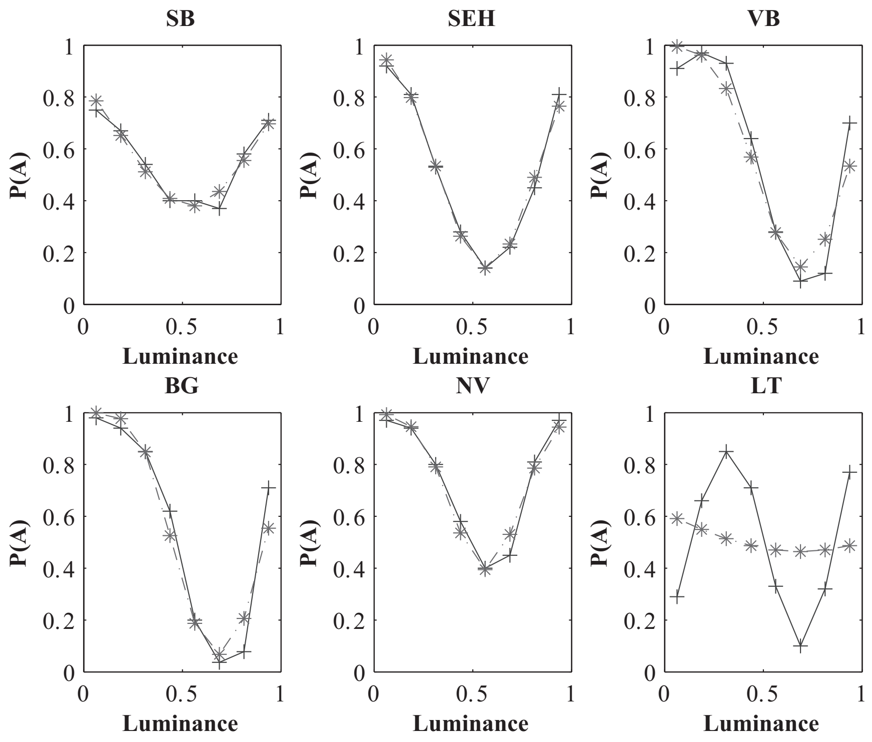

Figure 7.11 Proportion of 'A' responses predicted by GRT under ML parameter estimates.

may simply follow from the fact that it has more free parameters than both GCM and DEM. Similarly, DEM appears to give a better account of the data than GCM, but to what extent is this due to its extra parameter γ picking up nonsystematic residual variance?

Lines 68 to 72 prepare the results of the model fitting for model comparison by storing the results of each model's fit in a separate structure. Line 71 uses the eval function to assign the modeling results, collected in the temporary structure `t` in the preceding few lines, to a structure named either `GCM`, `GRT`, or `DEM`. Line 72 deletes a few variables because they differ in size between the different models and would return an error otherwise.

The last part of Listing 7.8 calls the function `infoCriteria` from Chapter 5 to obtain AIC and BIC values, AIC and BIC differences, and model weights for each participant. The AIC and BIC differences calculated by this code are shown in Table 7.7, and the corresponding model weights are shown in Table 7.8. For participants SB, SEH, BG, and NV, the model differences and model weights bear out the superior fit of the GRT suggested in the figures, with the model

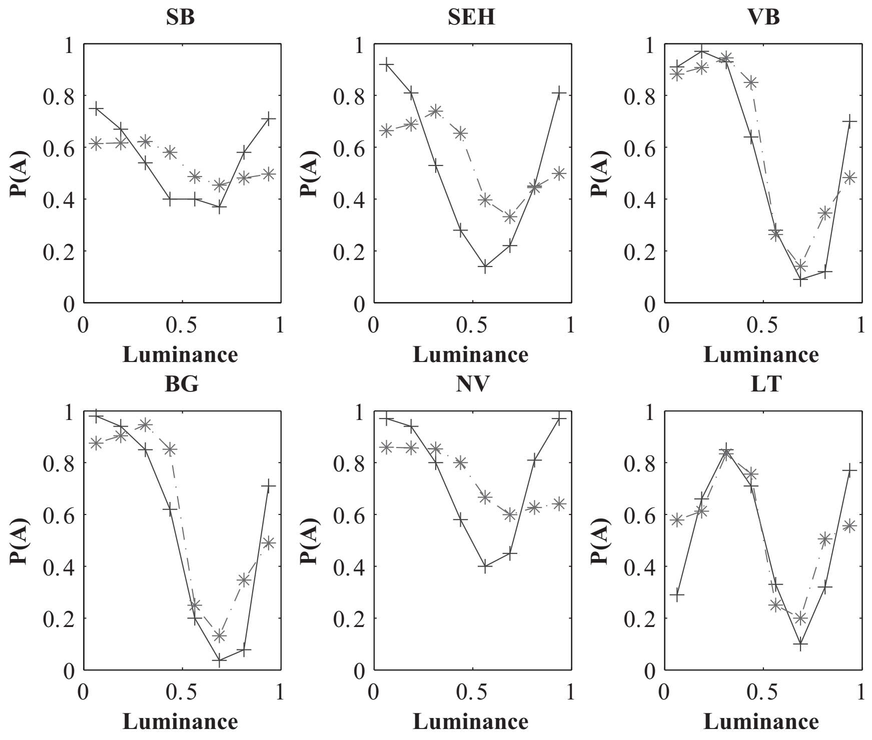

Figure 7.12 Proportion of 'A' responses predicted by DEM under ML parameter estimates.

weights for GRT indistinguishable from 1 for these participants. The information criterion results for VB are more informative. Recall that the fits for GRT and DEM looked similar in quality for VB. Tables 7.7 and 7.8 show that GRT nonetheless gives a statistically superior fit to VB's data, even when the extra parameter in the GRT is accounted for via the correction term in the AIC and BIC. Finally, Tables 7.7 and 7.8 show that LT's data are better fit by an exemplar model, particularly the DEM. Not only does DEM provide a superior fit to GRT in this case, but the information criteria point to DEM also being superior to GCM in its account of the data, indicating that the extra parameter γ is important for the exemplar model's account for this participant. Inspection of the ML parameter estimates in Table 7.6 is instructive for interpreting DEM's success. The row for LT shows that the estimated c was very large compared to the other participants and that the γ parameter was below 1. The large c for DEM (and, indeed, GCM, as shown in Table 7.4) means that stimuli only really match exemplars with the same luminance in memory, leading the model to strongly track the feedback probabilities. However, the $\gamma < 1$ means that the model undermatches those feedback

Table 7.7 AIC and BIC Differences Between the Three Models GCM, GRT, and DEM

		ΔAIC			Δ BIC	
	GCM	GRT	DEM	GCM	GRT	DEM
SB	178.42	0.00	180.38	166.49	0.00	174.42
SEH	715.39	0.00	715.83	703.46	0.00	709.87
VB	413.26	0.00	43.53	401.33	0.00	37.57
BG	703.67	0.00	337.16	691.74	0.00	331.20
NV	990.71	0.00	581.08	978.78	0.00	575.12
LT	18.78	527.95	0.00	12.82	533.92	0.00
sum	3020.23	527.95	1857.99	2954.61	533.92	1828.16

Note. Differences are calculated for each participant individually, and summed differences are shown in the bottom row.

Table 7.8 AIC and BIC Weights for the Three Models GCM, GRT, and DEM

		ΔAIC			Δ BIC	
	GCM	GRT	DEM	GCM	GRT	DEM
SB	0.00	1.00	0.00	0.00	1.00	0.00
SEH	0.00	1.00	0.00	0.00	1.00	0.00
VB	0.00	1.00	0.00	0.00	1.00	0.00
BG	0.00	1.00	0.00	0.00	1.00	0.00
NV	0.00	1.00	0.00	0.00	1.00	0.00
LT	0.00	0.00	1.00	0.00	0.00	1.00
mean	0.00	0.83	0.17	0.00	0.83	0.17

Note. Model weights are calculated for each participant individually, and mean weights are shown in the bottom row.

probabilities, which pulls the predicted proportion of ‘A’ responses back to the baseline of 0.5.

7.2.5 What Have We Learned About Categorization?

The results of the model fitting just presented lie heavily in favor of the GRT model. If we assume that all participants’ performance was based on a single model, we can examine the summed AIC/BIC differences in Table 7.7, or the average model weights in Table 7.8, and conclude that the data as a whole are most consistent with the GRT model, and by quite a margin. However, the data plotted in Figure 7.9, along with the modeling results in Tables 7.7 and 7.8, suggest that one participant, LT, is using a different set of representations or processes to perform this task. The most obvious conclusion based on the modeling results is that LT's performance is supported by matching to stored exemplars. However, that conclusion may be premature. Although the results correspond more to the ML predictions of GCM and DEM, notice that although LT's observed probability of giving an 'A' response changes dramatically between stimuli 1 and 2 (and, indeed, between 7 and 8), GCM and DEM's predictions change little between the same stimuli. Arguably, GCM and DEM are still failing to capture an important aspect of these data. Rouder and Ratcliff (2004) noted another possible explanation for these results: that LT was relying on boundaries (as in GRT) to categorize the stimuli but had inserted a third boundary somewhere around the lowest luminance stimulus despite this being a nonoptimal approach. Rouder and Ratcliff (2004) fit GRT with three boundaries to the data of LT and found it to give the best fit of all models and that it was able to produce the large change between stimuli 1 and 2 seen in LT's data. We leave implementation of that model as an exercise for you, but note that this reinforces the overall consistency between GRT and the data.

What does it mean to say that GRT is the "best" model? The main message is that under conditions of uncertainty, and with stimuli varying along a single dimension, people partition the stimulus space using boundaries and use those boundaries to categorize stimuli observed later on. We know it isn't simply that GRT assumes deterministic responding: The DEM model, an exemplar-based model with deterministic responding, failed to compete with the GRT in most cases.

One interesting note to sign off on is that hybrid models have become increasingly popular in accounting for categorization. For example, the RULEX model of Nosofsky and Palmeri (1998) assumes that people use rules to categorize stimuli but store category exceptions as exemplars. The COVIS model of Ashby et al. (1998) assumes that participants can use both explicit rules and implicit representations learned procedurally to categorize stimuli. Rouder and Ratcliff (2004) found that in some circumstances, their participants appeared to use exemplars to perform similar probabilistic categorization tasks and shifted between using boundaries and exemplars depending on the discriminability of the stimuli. This fits well with another hybrid model, ATRIUM (Erickson & Kruschke, 1998), in which rules and exemplars compete to categorize and learn each new object. ATRIUM has been shown to account for a variety of categorization experiments, particularly ones where participants appear to switch between strategies or mechanisms (e.g., Yang & Lewandowsky, 2004). Nonhybrid models of the types considered here are still useful for determining what type of model is predominantly in use in a particular task; nonetheless, a comprehensive model of categorization will need to explain how participants appear to be using quite different representations depending on the nature of the task and stimuli (e.g., Ashby & O'Brien, 2005; Erickson & Kruschke, 1998).

7.3 Conclusion

We have presented two detailed examples that build on the material from the preceding chapters. If you were able to follow those examples and understand what we did and why, then you have now gathered a very solid foundation in the techniques of computational and mathematical modeling.

Concerning future options for exploring the implementation and fitting of models, two directions for further study immediately come to mind: First, we suggest you explore Bayesian approaches to modeling and model selection. Bayesian techniques have recently become prominent in the field, and the material in this book forms a natural and solid foundation for further study in that area. Second, we suggest you investigate hierarchical (or multilevel) techniques. We touched on those techniques in Chapter 3 but were unable to explore them further in this volume. Many good introductions to both issues exist, and one particularly concise summary can be found in Shiffrin, Lee, Kim, and Wagenmakers (2008).

We next present a final chapter that moves away from the techniques of modeling and considers several general frameworks that have been adopted by modelers to explain psychological processes.

Notes

1. Note that our notation and symbology correspond to that used by Clare and Lewandowsky (2004) rather than by Clark (2003).

2. The fact that each replication involves different stimulus vectors (in addition to encoding of a different subset of features of the perpetrator) also implies that each replication involves a different set of faces in the lineup. This is desirable because it generalizes the simulation results across possible stimuli, but it does not reflect the procedure of most experiments in which a single lineup is used for all participants.

3. There are some minor differences between the simulations reported by Clare and Lewandowsky (2004) and those developed here, which were introduced to make the present example simple and transparent. The parameter estimates and model predictions reported here thus differ slightly (though not substantially) from those reported by Clare and Lewandowsky (2004).

4. In fact, we also select a method by which random numbers are selected, but we ignore this subtlety here for brevity.

5. If the number of replications is small, reseeding the random generator carries the risk that the model may capitalize on some chance idiosyncrasy in the random sequence. The

number of replications should therefore be ample (minimum of 100, preferably 1,000 or more).

6. We use *object* to refer to a collection of feature values; these features could be any of the dimensions defining a potentially highly multidimensional space, such as brightness, nose length, tone frequency, or time.

7. This follows from the assumption that members of a category are distributed as a multivariate Gaussian. The optimal boundary in such a case is a multidimensional plane (i.e., a manifold) tracing out those regions of space where the two adjacent categories are equally plausible, which for multivariate Gaussian distributions is mathematically described by a quadratic equation (Ashby, 1992a).

8. Alternatively, you can use the function `normcdf` in the MATLAB Statistics Toolbox.