

Miroslav Kubat

An Introduction to Machine Learning

Second Edition

 Springer

Miroslav Kubat

An Introduction to Machine Learning

Second Edition

 Springer

Miroslav Kubat
Department of Electrical and Computer Engineering
University of Miami
Coral Gables, FL, USA

ISBN 978-3-319-63912-3 ISBN 978-3-319-63913-0 (eBook)
DOI 10.1007/978-3-319-63913-0

Library of Congress Control Number: 2017949183

© Springer International Publishing AG 2015, 2017

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Contents

1	A Simple Machine-Learning Task	1
1.1	Training Sets and Classifiers	1
1.2	Minor Digression: Hill-Climbing Search	5
1.3	Hill Climbing in Machine Learning	8
1.4	The Induced Classifier's Performance	11
1.5	Some Difficulties with Available Data	13
1.6	Summary and Historical Remarks	15
1.7	Solidify Your Knowledge	16
2	Probabilities: Bayesian Classifiers	19
2.1	The Single-Attribute Case	19
2.2	Vectors of Discrete Attributes	22
2.3	Probabilities of Rare Events: Exploiting the Expert's Intuition	26
2.4	How to Handle Continuous Attributes	30
2.5	Gaussian "Bell" Function: A Standard <i>pdf</i>	33
2.6	Approximating PDFs with Sets of Gaussians	34
2.7	Summary and Historical Remarks	36
2.8	Solidify Your Knowledge	40
3	Similarities: Nearest-Neighbor Classifiers	43
3.1	The <i>k</i> -Nearest-Neighbor Rule	43
3.2	Measuring Similarity	46
3.3	Irrelevant Attributes and Scaling Problems	49
3.4	Performance Considerations	52
3.5	Weighted Nearest Neighbors	55
3.6	Removing Dangerous Examples	57
3.7	Removing Redundant Examples	59
3.8	Summary and Historical Remarks	61
3.9	Solidify Your Knowledge	62

4	Inter-Class Boundaries: Linear and Polynomial Classifiers	65
4.1	The Essence	65
4.2	The Additive Rule: Perceptron Learning	69
4.3	The Multiplicative Rule: WINNOW	73
4.4	Domains with More Than Two Classes	76
4.5	Polynomial Classifiers	79
4.6	Specific Aspects of Polynomial Classifiers	81
4.7	Numerical Domains and Support Vector Machines	84
4.8	Summary and Historical Remarks	86
4.9	Solidify Your Knowledge	87
5	Artificial Neural Networks	91
5.1	Multilayer Perceptrons as Classifiers	91
5.2	Neural Network's Error	95
5.3	Backpropagation of Error	97
5.4	Special Aspects of Multilayer Perceptrons	100
5.5	Architectural Issues	104
5.6	Radial-Basis Function Networks	106
5.7	Summary and Historical Remarks	109
5.8	Solidify Your Knowledge	110
6	Decision Trees	113
6.1	Decision Trees as Classifiers	113
6.2	Induction of Decision Trees	117
6.3	How Much Information Does an Attribute Convey?	119
6.4	Binary Split of a Numeric Attribute	122
6.5	Pruning	126
6.6	Converting the Decision Tree into Rules	130
6.7	Summary and Historical Remarks	132
6.8	Solidify Your Knowledge	133
7	Computational Learning Theory	137
7.1	PAC Learning	137
7.2	Examples of PAC Learnability	141
7.3	Some Practical and Theoretical Consequences	143
7.4	VC-Dimension and Learnability	145
7.5	Summary and Historical Remarks	148
7.6	Exercises and Thought Experiments	149
8	A Few Instructive Applications	151
8.1	Character Recognition	151
8.2	Oil-Spill Recognition	155
8.3	Sleep Classification	158
8.4	Brain-Computer Interface	161
8.5	Medical Diagnosis	165

8.6	Text Classification	167
8.7	Summary and Historical Remarks	169
8.8	Exercises and Thought Experiments	170
9	Induction of Voting Assemblies	173
9.1	Bagging	173
9.2	Schapire's Boosting	176
9.3	Adaboost: Practical Version of Boosting	179
9.4	Variations on the Boosting Theme	183
9.5	Cost-Saving Benefits of the Approach	185
9.6	Summary and Historical Remarks	187
9.7	Solidify Your Knowledge	188
10	Some Practical Aspects to Know About	191
10.1	A Learner's Bias	191
10.2	Imbalanced Training Sets	194
10.3	Context-Dependent Domains	199
10.4	Unknown Attribute Values	202
10.5	Attribute Selection	204
10.6	Miscellaneous	206
10.7	Summary and Historical Remarks	208
10.8	Solidify Your Knowledge	208
11	Performance Evaluation	211
11.1	Basic Performance Criteria	211
11.2	Precision and Recall	214
11.3	Other Ways to Measure Performance	219
11.4	Learning Curves and Computational Costs	222
11.5	Methodologies of Experimental Evaluation	224
11.6	Summary and Historical Remarks	227
11.7	Solidify Your Knowledge	228
12	Statistical Significance	231
12.1	Sampling a Population	231
12.2	Benefiting from the Normal Distribution	235
12.3	Confidence Intervals	239
12.4	Statistical Evaluation of a Classifier	241
12.5	Another Kind of Statistical Evaluation	244
12.6	Comparing Machine-Learning Techniques	245
12.7	Summary and Historical Remarks	247
12.8	Solidify Your Knowledge	248
13	Induction in Multi-Label Domains	251
13.1	Classical Machine Learning in Multi-Label Domains	251
13.2	Treating Each Class Separately: Binary Relevance	254
13.3	Classifier Chains	256

13.4	Another Possibility: Stacking	258
13.5	A Note on Hierarchically Ordered Classes.....	260
13.6	Aggregating the Classes	263
13.7	Criteria for Performance Evaluation.....	265
13.8	Summary and Historical Remarks.....	268
13.9	Solidify Your Knowledge	269
14	Unsupervised Learning	273
14.1	Cluster Analysis	273
14.2	A Simple Algorithm: <i>k</i> -Means.....	277
14.3	More Advanced Versions of <i>k</i> -Means	281
14.4	Hierarchical Aggregation	283
14.5	Self-Organizing Feature Maps: Introduction.....	286
14.6	Some Important Details	289
14.7	Why Feature Maps?	291
14.8	Summary and Historical Remarks.....	293
14.9	Solidify Your Knowledge	294
15	Classifiers in the Form of Rulesets.....	297
15.1	A Class Described By Rules	297
15.2	Inducing Rulesets by Sequential Covering.....	300
15.3	Predicates and Recursion	302
15.4	More Advanced Search Operators.....	305
15.5	Summary and Historical Remarks.....	306
15.6	Solidify Your Knowledge	307
16	The Genetic Algorithm.....	309
16.1	The Baseline Genetic Algorithm	309
16.2	Implementing the Individual Modules	311
16.3	Why It Works.....	314
16.4	The Danger of Premature Degeneration.....	317
16.5	Other Genetic Operators	319
16.6	Some Advanced Versions	321
16.7	Selections in <i>k</i> -NN Classifiers	324
16.8	Summary and Historical Remarks.....	327
16.9	Solidify Your Knowledge	328
17	Reinforcement Learning	331
17.1	How to Choose the Most Rewarding Action.....	331
17.2	States and Actions in a Game.....	334
17.3	The SARSA Approach	337
17.4	Summary and Historical Remarks.....	338
17.5	Solidify Your Knowledge	338
	Bibliography	341
	Index.....	347

Introduction

Machine learning has come of age. And just in case you might think this is a mere platitude, let me clarify.

The dream that machines would one day be able to learn is as old as computers themselves, perhaps older still. For a long time, however, it remained just that: a dream. True, Rosenblatt’s perceptron did trigger a wave of activity, but in retrospect, the excitement has to be deemed short-lived. As for the attempts that followed, these fared even worse; barely noticed, often ignored, they never made a breakthrough—no software companies, no major follow-up research, and not much support from funding agencies. Machine learning remained an underdog, condemned to live in the shadow of more successful disciplines. The grand ambition lay dormant.

And then it all changed.

A group of visionaries pointed out a weak spot in the knowledge-based systems that were all the rage in the 1970s’ artificial intelligence: where was the “knowledge” to come from? The prevailing wisdom of the day insisted that it should take the form of *if-then* rules put together by the joint effort of engineers and field experts. Practical experience, though, was unconvincing. Experts found it difficult to communicate what they knew to engineers. Engineers, in turn, were at a loss as to what questions to ask and what to make of the answers. A few widely publicized success stories notwithstanding, most attempts to create a knowledge base of, say, tens of thousands of such rules proved frustrating.

The proposition made by the visionaries was both simple and audacious. If it is so hard to tell a machine exactly how to go about a certain problem, why not provide the instruction indirectly, conveying the necessary skills by way of examples from which the computer will—yes—*learn!*

Of course, this only makes sense if we can rely on the existence of algorithms to do the learning. This was the main difficulty. As it turned out, neither Rosenblatt’s perceptron nor the techniques developed after it were very useful. But the absence of the requisite machine-learning techniques was not an obstacle; rather, it was a challenge that inspired quite a few brilliant minds. The idea of endowing computers with learning skills opened new horizons and created a large amount of excitement. The world was beginning to take notice.

The bombshell exploded in 1983. *Machine Learning: The AI Approach*¹ was a thick volume of research papers which proposed the most diverse ways of addressing the great mystery. Under their influence, a new scientific discipline was born—virtually overnight. Three years later, a follow-up book appeared and then another. A soon-to-become-prestigious scientific journal was founded. Annual conferences of great repute were launched. And dozens, perhaps hundreds, of doctoral dissertations, were submitted and successfully defended.

In this early stage, the question was not only *how* to learn but also *what* to learn and *why*. In retrospect, those were wonderful times, so creative that they deserve to be remembered with nostalgia. It is only to be regretted that so many great thoughts later came to be abandoned. Practical needs of realistic applications got the upper hand, pointing to the most promising avenues for further efforts. After a period of enchantment, concrete research strands crystallized: induction of the *if-then* rules for knowledge-based systems; induction of classifiers, programs capable of improving their skills based on experience; automatic fine-tuning of Prolog programs; and some others. So many were the directions that some leading personalities felt it necessary to try to steer further development by writing monographs, some successful, others less so.

An important watershed was Tom Mitchell's legendary textbook.² This summarized the state of the art of the field in a format appropriate for doctoral students and scientists alike. One by one, universities started offering graduate courses that were usually built around this book. Meanwhile, the research methodology became more systematic, too. A rich repository of machine-learning test beds was created, making it possible to compare the performance of learning algorithms. Statistical methods of evaluation became widespread. Public domain versions of most popular programs were made available. The number of scientists dealing with this discipline grew to thousands, perhaps even more.

Now, we have reached the stage where a great many universities are offering machine learning as an undergraduate class. This is quite a new situation. As a rule, these classes call for a different kind of textbook. Apart from mastering the baseline techniques, future engineers need to develop a good grasp of the strengths and weaknesses of alternative approaches; they should be aware of the peculiarities and idiosyncrasies of different paradigms. Above all, they must understand the circumstances under which some techniques succeed and others fail. Only then will they be able to make the right choices when addressing concrete applications. A textbook that is to provide all of the above should contain less mathematics, but a lot of practical advice.

These then are the considerations that have dictated the size, structure, and style of a teaching text meant to provide the material for a one-semester introductory course.

¹Edited by R. Michalski, J. Carbonell, and T. Mitchell.

²T. Mitchell, *Machine Learning*, McGraw-Hill (1997).

The first problem is the choice of material. At a time when high-tech companies are establishing machine-learning groups, universities have to provide the students with such knowledge, skills, and understanding that are relevant to the current needs of the industry. For this reason, preference has been given to Bayesian classifiers, nearest-neighbor classifiers, linear and polynomial classifiers, decision trees, the fundamentals of the neural networks, and the principle of the boosting algorithms. Significant space has been devoted to certain typical aspects of concrete engineering applications. When applied to really difficult tasks, the baseline techniques are known to behave not exactly the same way they do in the toy domains employed by the instructor. One has to know what to expect.

The book consists of 17 chapters, each covering one major topic. The chapters are divided into sections, each devoted to one critical problem. The student is advised to proceed to the next section only after having answered the set of 2–4 “control questions” at the end of the previous section. These questions are here to help the student decide whether he or she has mastered the given material. If not, it is necessary to return to the previous text.

As they say, only practice makes perfect. This is why at the end of each chapter are exercises to encourage the necessary practicing. Deeper insight into the diverse aspects of the material will then be gained by going through the thought experiments that follow. These are more difficult, but it is only through hard work that an engineer develops the right kind of understanding. The acquired knowledge is then further solidified by suggested computer projects. Programming is important, too. Nowadays, everybody is used to downloading the requisite software from the web. This shortcut, however, is not recommended to the student of this book. It is only by being forced to flesh out all the details of a computer program that you learn to appreciate all the subtle points of the machine-learning techniques presented here.

Chapter 1

A Simple Machine-Learning Task

You will find it difficult to describe your mother's face accurately enough for your friend to recognize her in a supermarket. But if you show him a few of her photos, he will immediately spot the tell-tale traits he needs. As they say, a picture—an example—is worth a thousand words.

This is what we want our technology to emulate. Unable to define certain objects or concepts with adequate accuracy, we want to convey them to the machine by way of examples. For this to work, however, the computer has to be able to convert the examples into knowledge. Hence our interest in algorithms and techniques for *machine learning*, the topic of this textbook.

The first chapter formulates the task as a search problem, introducing hill-climbing search not only as our preliminary attempt to address the machine-learning task, but also as a tool that will come handy in a few auxiliary problems to be encountered in later chapters. Having thus established the foundation, we will proceed to such issues as performance criteria, experimental methodology, and certain aspects that make the learning process difficult—and interesting.

1.1 Training Sets and Classifiers

Let us introduce the problem, and certain fundamental concepts that will accompany us throughout the rest of the book.

The Set of Pre-Classified Training Examples Figure 1.1 shows six pies that Johnny likes, and six that he does not. These *positive* and *negative examples* of the underlying concept constitute a *training set* from which the machine is to induce a *classifier*—an algorithm capable of categorizing any future pie into one of the two *classes*: positive and negative.

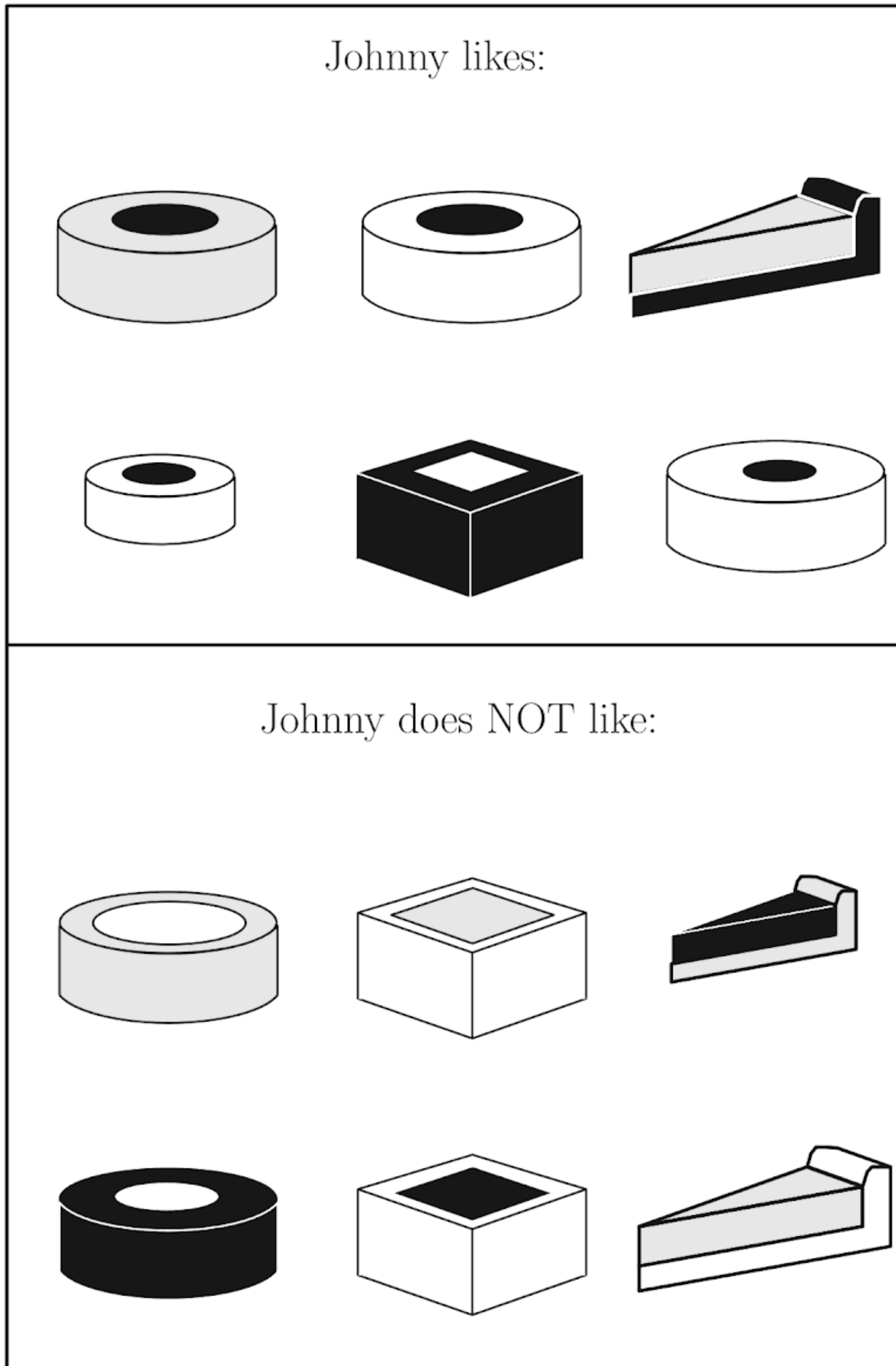


Fig. 1.1 A simple machine-learning task: induce a classifier capable of labeling future pies as positive and negative instances of “a pie that Johnny likes”

The number of classes can of course be greater. Thus a classifier that decides whether a landscape snapshot was taken in *spring*, *summer*, *fall*, or *winter* distinguishes four. Software that identifies characters scribbled on an *iPad* needs at least 36 classes: 26 for letters and 10 for digits. And document-categorization systems are capable of identifying hundreds, even thousands of different topics. Our only motivation for choosing a two-class domain is its simplicity.

Table 1.1 The twelve training examples expressed in a matrix form

Example	Shape	Crust		Filling		Class
		Size	Shade	Size	Shade	
ex1	Circle	Thick	Gray	Thick	Dark	pos
ex2	Circle	Thick	White	Thick	Dark	pos
ex3	Triangle	Thick	Dark	Thick	Gray	pos
ex4	Circle	Thin	White	Thin	Dark	pos
ex5	Square	Thick	Dark	Thin	White	pos
ex6	Circle	Thick	White	Thin	Dark	pos
ex7	Circle	Thick	Gray	Thick	White	neg
ex8	Square	Thick	White	Thick	Gray	neg
ex9	Triangle	Thin	Gray	Thin	Dark	neg
ex10	Circle	Thick	Dark	Thick	White	neg
ex11	Square	Thick	White	Thick	Dark	neg
ex12	Triangle	Thick	White	Thick	Gray	neg

Attribute Vectors To be able to communicate the training examples to the machine, we have to describe them in an appropriate way. The most common mechanism relies on so-called *attributes*. In the “pies” domain, five may be suggested: shape (circle, triangle, and square), crust-size (thin or thick), crust-shade (white, gray, or dark), filling-size (thin or thick), and filling-shade (white, gray, or dark). Table 1.1 specifies the values of these attributes for the twelve examples in Fig. 1.1. For instance, the pie in the upper-left corner of the picture (the table calls it ex1) is described by the following conjunction:

```
(shape=circle) AND (crust-size=thick) AND (crust-shade=gray)
AND (filling-size=thick) AND (filling-shade=dark)
```

A Classifier to Be Induced The training set constitutes the input from which we are to induce the classifier. But *what* classifier?

Suppose we want it in the form of a boolean function that is *true* for positive examples and *false* for negative ones. Checking the expression `[(shape=circle) AND (filling-shade=dark)]` against the training set, we can see that its value is *false* for all negative examples: while it is possible to find negative examples that are circular, none of these has a dark filling. As for the positive examples, however, the expression is *true* for four of them and *false* for the remaining two. This means that the classifier makes two errors, a transgression we might refuse to tolerate, suspecting there is a better solution. Indeed, the reader will easily verify that the following expression never goes wrong on the entire training set:

```
[ (shape=circle) AND (filling-shade=dark) ] OR
[ NOT(shape=circle) AND (crust-shade=dark) ]
```


Problems with a Brute-Force Approach How does a machine find a classifier of this kind? Brute force (something that computers are so good at) will not do here. Just consider how many different examples can be distinguished by the given set of attributes in the “pies” domain. For each of the three different shapes, there are two alternative `crust-sizes`, the number of combinations being $3 \times 2 = 6$. For each of these, the next attribute, `crust-shade`, can acquire three different values, which brings the number of combinations to $3 \times 2 \times 3 = 18$. Extending this line of reasoning to *all* attributes, we realize that the size of the *instance space* is $3 \times 2 \times 3 \times 2 \times 3 = 108$ different examples.

Each subset of these examples—and there are 2^{108} subsets!—may constitute the list of positive examples of someone’s notion of a “good pie.” And each such subset can be characterized by at least one boolean expression. Running each of these classifiers through the training set is clearly out of the question.

Manual Approach and Search Uncertain about how to invent a classifier-inducing algorithm, we may try to glean some inspiration from an attempt to create a classifier “manually,” by the good old-fashioned pencil-and-paper method. When doing so, we begin with some tentative initial version, say, `shape=circular`. Having checked it against the training set, we find it to be *true* for four positive examples, but also for two negative ones. Apparently, the classifier needs to be “narrowed” (specialized) so as to exclude the two negative examples. One way to go about the specialization is to add a conjunction, such as when turning `shape=circular` into `[(shape=circular) AND (filling-shade=dark)]`. This new expression, while *false* for all negative examples, is still imperfect because it covers only four (`ex1`, `ex2`, `ex4`, and `ex6`) of the six positive examples. The next step should therefore attempt some generalization, perhaps by adding a disjunction: `{ [(shape=circular) AND (filling-shade=dark)] OR (crust-size=thick) }`. We continue in this way until we find a 100% accurate classifier (if it exists).

The lesson from this little introspection is that the classifier can be created by means of a sequence of specialization and generalization steps which gradually modify a given version of the classifier until it satisfies certain predefined requirements. This is encouraging. Readers with background in Artificial Intelligence will recognize this procedure as a *search* through the space of boolean expressions. And Artificial Intelligence is known to have developed and explored quite a few of search algorithms. It may be an idea to take a look at least at one of them.

What Have You Learned?

To make sure you understand the topic, try to answer the following questions. If needed, return to the appropriate place in the text.

- What is the input and output of the learning problem we have just described?
- How do we describe the training examples? What is *instance space*? Can we calculate its size?

- In the “pies” domain, find a boolean expression that correctly classifies all the training examples from Table 1.1.

1.2 Minor Digression: Hill-Climbing Search

Let us now formalize what we mean by *search*, and then introduce one popular algorithm, the so-called *hill climbing*. Artificial Intelligence defines *search* something like this: starting from an *initial state*, find a sequence of steps which, proceeding through a set of interim *search states*, lead to a predefined *final state*. The individual steps—transitions from one search state to another—are carried out by *search operators* which, too, have been pre-specified by the programmer. The order in which the search operators are applied follows a specific *search strategy* (Fig. 1.2).

Hill Climbing: An Illustration One popular search strategy is *hill climbing*. Let us illustrate its essence on a well-known brain-teaser, the sliding-tiles puzzle. The board of a trivial version of this game consists of nine squares arranged in three rows, eight covered by numbered tiles (integers from 1 to 8), the last left empty. We convert one search state into another by sliding to the empty square a tile from one of its neighbors. The goal is to achieve a pre-specified arrangement of the tiles.

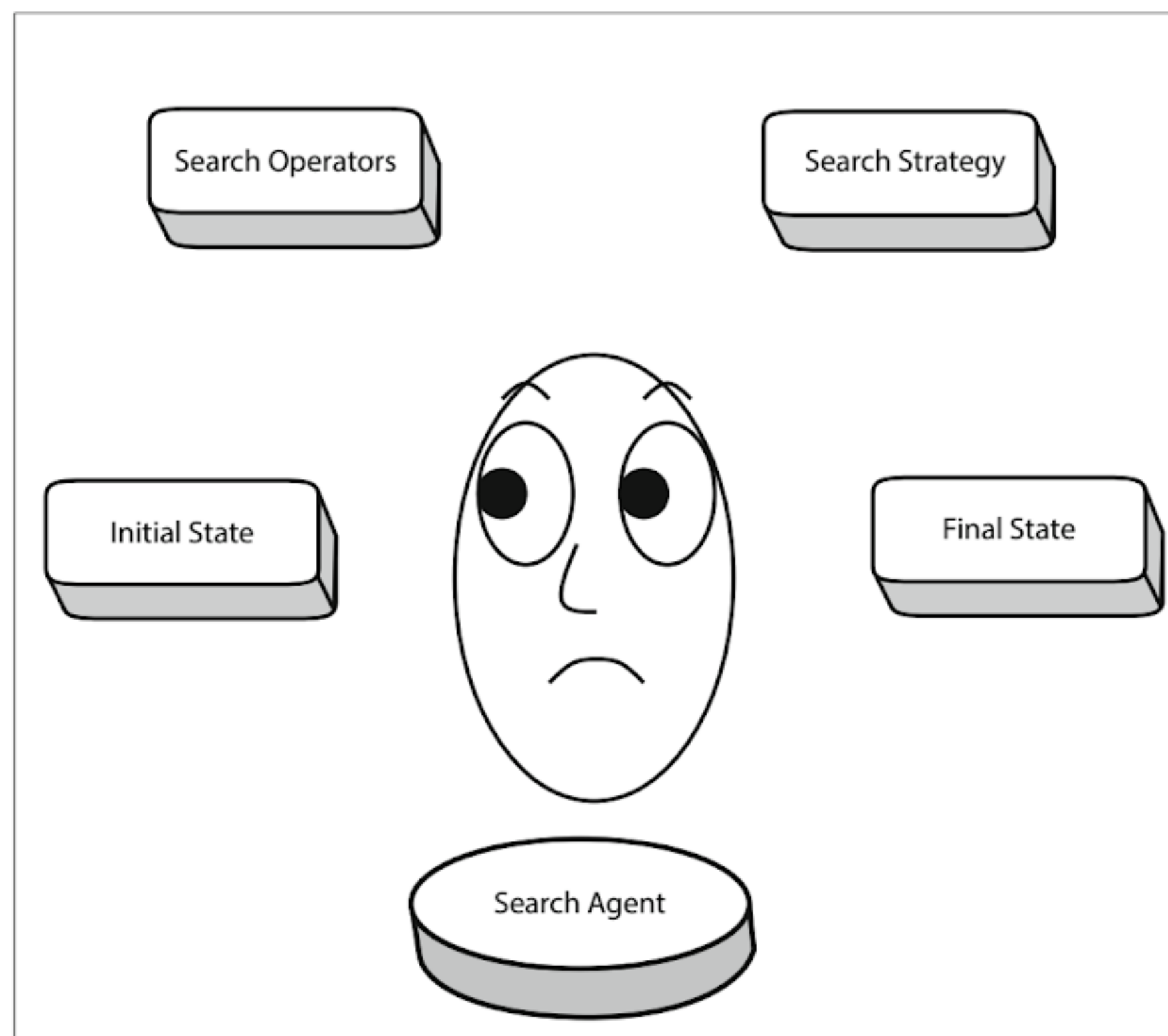


Fig. 1.2 A search problem is characterized by an initial state, final state, search operators, and a search strategy

The flowchart in Fig. 1.3 starts with a concrete initial state, in which we can choose between two operators: “move `tile-6` up” and “move `tile-2` to the left.” The choice is guided by an *evaluation function* that estimates for each state its distance from the goal. A simple possibility is to count the squares that the tiles have to traverse before reaching their final destinations. In the initial state, tiles 2, 4, and 5 are already in the right locations; tile 3 has to be moved by four squares; and each of the tiles 1, 6, 7, and 8 have to be moved by two squares. This sums up to distance $d = 4 + 4 \times 2 = 12$.

In Fig. 1.3, each of the two operators applicable to the initial state leads to a state whose distance from the final state is $d = 13$. In the absence of any other guidance, we choose randomly and go to the left, reaching the situation where the empty square is in the middle of the top row. Here, three moves are possible. One of them would only get us back to the initial state, and can thus be ignored; as for the remaining two, one results in a state with $d = 14$, the other in a state with $d = 12$. The latter being the lower value, this is where we go. The next step is trivial because only one move gets us to a state that has not been visited before. After this, we again face the choice between two alternatives . . . and this how the search continues until it reaches the final state.

Alternative Termination Criteria and Evaluation Functions Other *termination criteria* can be considered, too. The search can be instructed to stop when the maximum allotted time has elapsed (we do not want the computer to run forever), when the number of visited states has exceeded a certain limit, when something sufficiently close to the final state has been found, when we have realized that all states have already been visited, and so on, the concrete formulation reflecting critical aspects of the given application, sometimes combining two or more criteria in one.

By the way, the evaluation function employed in the sliding-tiles example was fairly simple, barely accomplishing its mission: to let the user convey some notion of his or her understanding of the problem, to provide a hint as to which move a human solver might prefer. To succeed in a realistic application, we would have to come up with a more sophisticated function. Quite often, *many* different alternatives can be devised, each engendering a different sequence of steps. Some will be quick in reaching the solution, others will follow a more circuitous path. The program’s performance will then depend on the programmer’s ability to pick the right one.

The Algorithm of Hill Combing The algorithm is summarized by the pseudocode in Table 1.2. Details will of course depend on each individual’s programming style, but the code will almost always contain a few typical functions. One of them compares two states and returns *true* if they are identical; this is how the program ascertains that the final state has been reached. Another function takes a given search state and applies to it all search operators, thus creating a complete set of “child states.” To avoid infinite loops, a third function checks whether a state has already been investigated. A fourth calculates for a given state its distance from the final

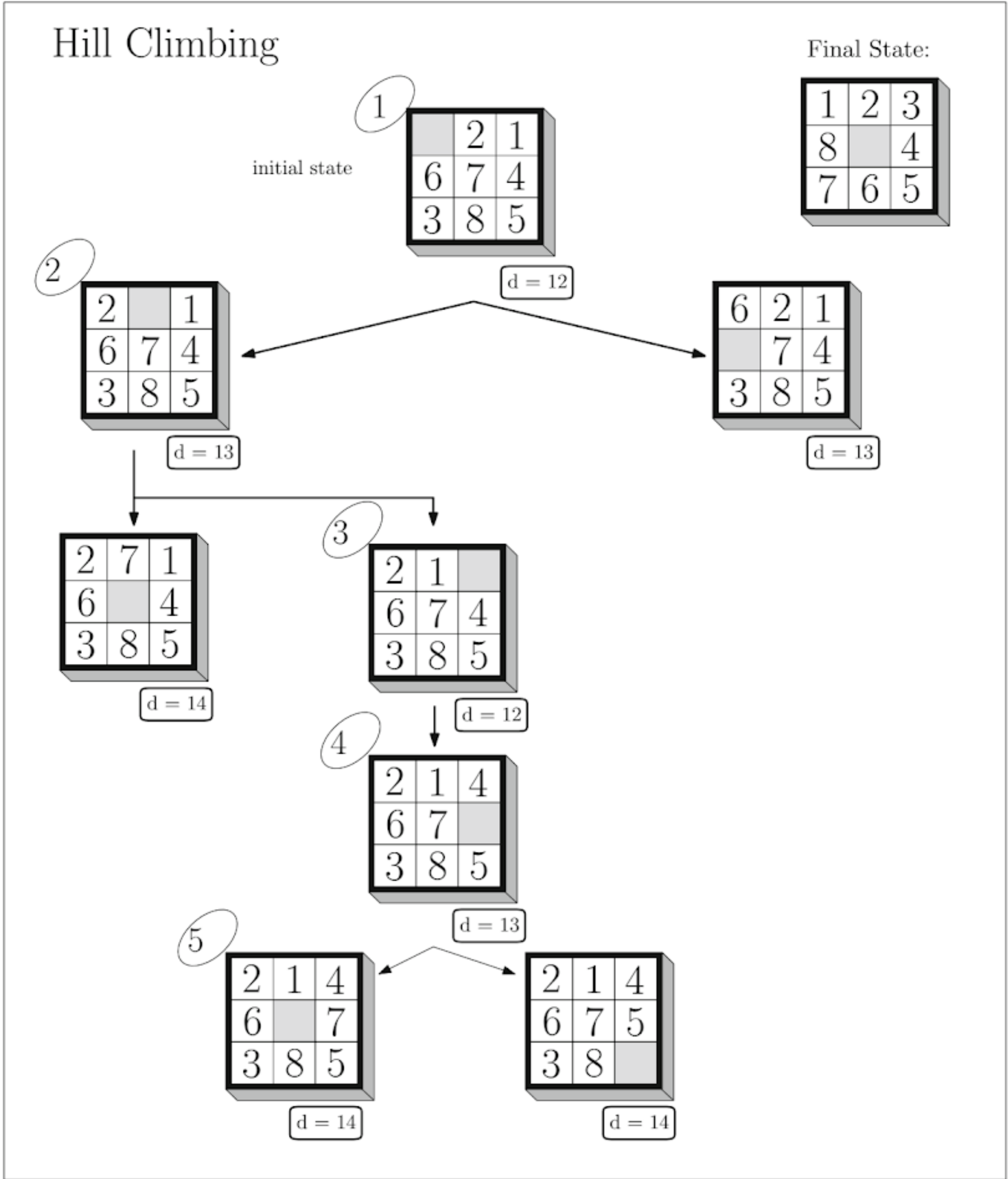


Fig. 1.3 Hill climbing. Circled integers indicate the order in which the search states are visited. d is a state’s distance from the final state as calculated by the given evaluation function. Ties are broken randomly

state, and a fifth sorts the “child” states according to the distances thus calculated and places them at the front of the list L . And the last function checks if a termination criterion has been satisfied.¹

One last observation: at some of the states in Fig. 1.3, no “child” offers any improvement over its “parent,” a lower d -value being achieved only after temporary compromises. This is what a mountain climber may experience, too: sometimes, he has to traverse a valley before being able to resume the ascent. The mountain-climbing metaphor, by the way, is what gave this technique its name.

¹For simplicity, the pseudocode ignores termination criteria other than reaching, or failing to reach, the final state.

Table 1.2 Hill-climbing search algorithm

-
1. Create two lists, L and L_{seen} . At the beginning, L contains only the initial state, and L_{seen} is empty.
 2. Let n be the first element of L . Compare this state with the final state. If they are identical, stop with success.
 3. Apply to n all available search operators, thus obtaining a set of new states. Discard those states that already exist in L_{seen} . As for the rest, sort them by the evaluation function and place them at the front of L .
 4. Transfer n from L into the list, L_{seen} , of the states that have been investigated.
 5. If $L = \emptyset$, stop and report failure. Otherwise, go to 2.
-

What Have You Learned?

To make sure you understand the topic, try to answer the following questions. If needed, return to the appropriate place in the text.

- How does Artificial Intelligence define the search problem? What do we understand under the terms, “search space” and “search operators”?
- What is the role of the evaluation function? How does it affect the hill-climbing behavior?

1.3 Hill Climbing in Machine Learning

We are ready to explore the concrete ways of applying hill climbing to the needs of machine learning.

Hill Climbing and Johnny’s Pies Let us begin with the problem of how to decide which pies Johnny likes. The input consists of a set of training examples, each described by the available attributes. The output—the *final state*—is a boolean expression that is *true* for each positive example in the training set, and *false* for each negative example. The expression involves attribute-value pairs, logical operators (conjunction, disjunction, and negation), and such combination of parentheses as may be needed. The evaluation function measures the given expression’s error rate on the training set. For the *initial state*, any randomly generated expression can be used. In Fig. 1.4, we chose `(shape=circle)`, on the grounds that more than a half of the training examples are circular.

As for the *search operator*, one possibility is to add a conjunction as illustrated in the upper part of Fig. 1.4: for instance, the root’s leftmost child is obtained by replacing `(shape=circle)` with `[(shape=circle) AND (filling-shade=dark)]` (in the picture, logical AND is represented by the symbol “ \wedge ”). Note how many different expressions this operator generates even in our toy domain. To `shape=circle`, any other attribute-value pair can be

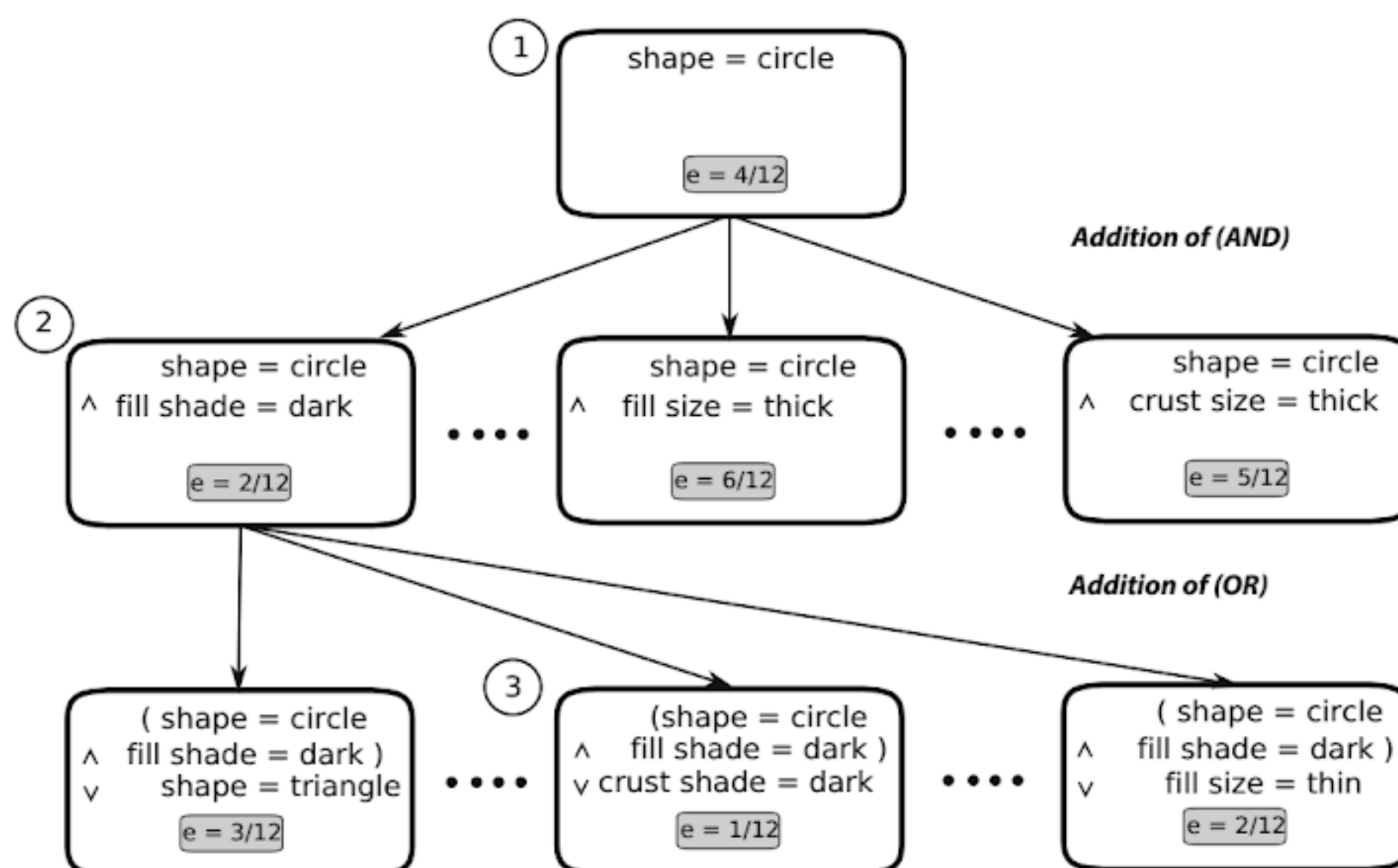


Fig. 1.4 Hill-climbing search in the “pies” domain

“ANDed.” Since the remaining four attributes (apart from `shape`) acquire 2, 3, 2, and 3 different values, respectively, the total number of terms that can be added to `(shape=circle)` is $2 \times 2 \times 3 = 36$.²

Alternatively, we may choose to add a disjunction, as illustrated (in the picture) by the three expansions of the leftmost child. Other operators may “remove a conjunct,” “remove a disjunct,” “add a negation,” “negate a term,” various ways of manipulating parentheses, and so on. All in all, hundreds of search operators can be applied to each state, and then again to the resulting states. This can be hard to manage even in this very simple domain.

Numeric Attributes In the “pies” domain, each attribute acquires one out of a few discrete values, but in realistic applications, some attributes will probably be numeric. For instance, each pie has a `price`, an attribute whose values come from a continuous domain. What will the search look like then?

To keep things simple, suppose there are only two attributes: `weight` and `price`. This limitation makes it possible, in Fig. 1.5, to represent each training example by a point in a plane. The reader can see that examples belonging to the same class tend to occupy a specific region, and curves separating individual regions can be defined—expressed mathematically as lines, circles, polynomials. For instance, the right part of Fig. 1.5 shows three different circles, each of which can act as a classifier: examples inside the circle are deemed positive; those outside, negative. Again, some of these classifiers are better than others. How will hill climbing go about finding the best ones? Here is one possibility.

²Of the 36 new states thus created, Fig. 1.4 shows only three.

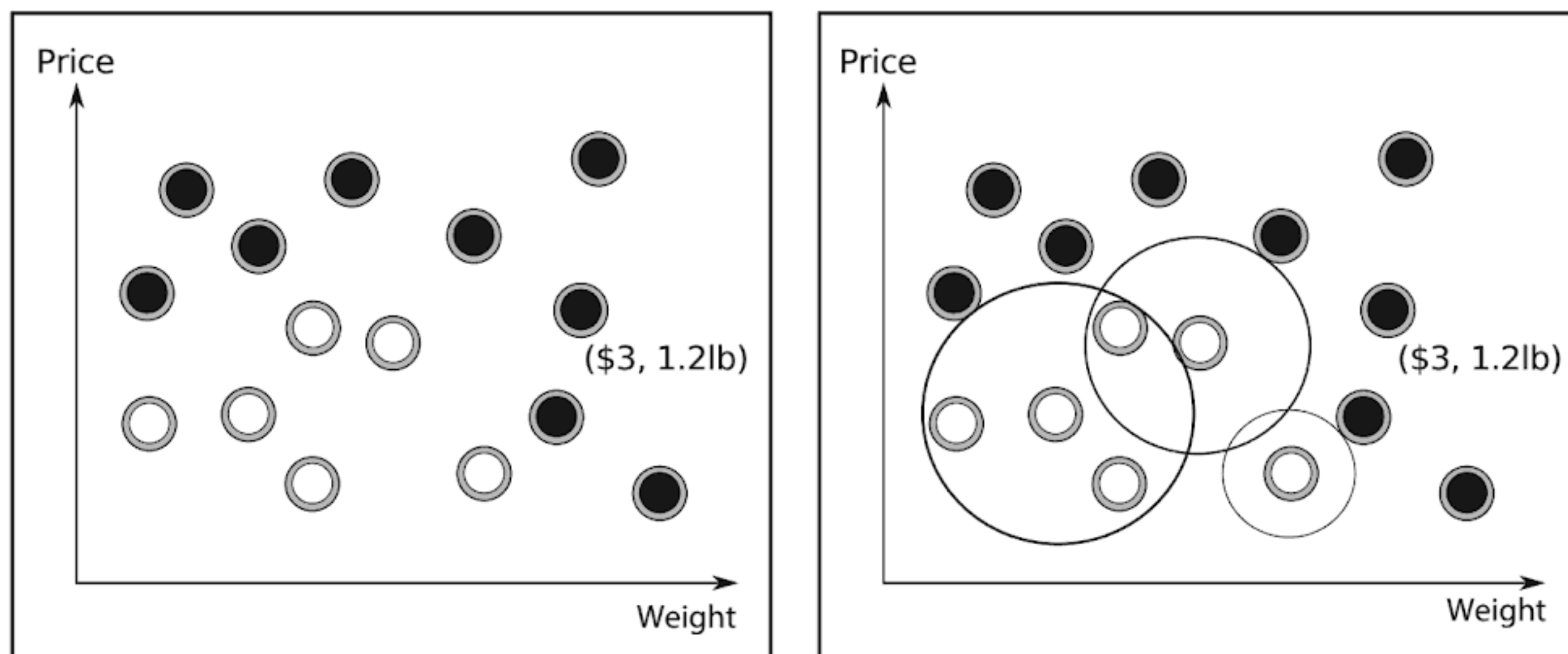


Fig. 1.5 *On the left:* a domain with continuous attributes; *on the right:* some “circular” classifiers

Hill Climbing in a Domain with Numeric Attributes

Initial State A circle is defined by its center and radius. We can identify the initial center with a randomly selected positive example, making the initial radius so small that the circle contains only this single example.

Search Operators Two search operators can be used: one increases the circle’s radius, and the other shifts the center from one training example to another. In the former, we also have to determine *how much* the radius should change. One idea is to increase it only so much as to make the circle encompass one additional training example. At the beginning, only one training example is inside. After the first step, there will be two, then three, four, and so on.

Final State The circle may not be an ideal figure to represent the positive region. In this event, a 100% accuracy may not be achievable, and we may prefer to define the final state as, say, a “classifier that correctly classifies 95% of the training examples.”

Evaluation Function As before, we choose to minimize the error rate.

What Have You Learned?

To make sure you understand the topic, try to answer the following questions. If needed, return to the appropriate place in the text.

- What aspects of search must be specified before we can employ hill climbing in machine learning?
- What search operators can be used in the “pies” domain and what in the “circles” domain? How can we define the evaluation function, the initial state, and the final state?

1.4 The Induced Classifier's Performance

So far, we have measured the error rate by comparing the training examples' known classes with those recommended by the classifier. Practically speaking, though, our goal is *not* to re-classify objects whose classes we already know; what we really want is to label *future examples*, those of whose classes we are as yet ignorant. The classifier's anticipated performance on these is estimated experimentally. It is important to know how.

Independent Testing Examples The simplest scenario will divide the available pre-classified examples into two parts: the training set, from which the classifier is induced, and the *testing set*, on which it is evaluated (Fig. 1.6). Thus in the “pies” domain, with its 12 pre-classified examples, the induction may be carried out on randomly selected eight, and the testing on the remaining four. If the classifier then “guesses” correctly the class of three testing examples (while going wrong on one), its performance is estimated as 75%.

Reasonable though this approach may appear, it suffers from a major drawback: a random choice of eight training examples may not be sufficiently representative of the underlying concept—and the same applies to the (even smaller) testing set. If we induce the meaning of a mammal from a training set consisting of a whale, a dolphin, and a platypus, the learner may be led to believe that mammals live in the sea (whale, dolphin), and sometimes lay eggs (platypus), hardly an opinion a biologist will embrace. And yet, another choice of training examples may result in a

Fig. 1.6 Pre-classified examples are divided into the training and testing sets

available examples	
training set	testing set

classifier satisfying the highest standards. The point is, a different training/testing set division gives rise to a different classifier—and also to a different estimate of future performance. This is particularly serious if the number of pre-classified examples is small.

Suppose we want to compare two machine learning algorithms in terms of the quality of the products they induce. The problem of non-representative training sets can be mitigated by so-called *random subsampling*.³ The idea is to repeat the random division into the training and testing sets several times, always inducing a classifier from the *i*-th training set, and then measuring the error rate, E_i , on the *i*-th testing set. The algorithm that delivers classifiers with the lower average value of E_i 's is deemed better—as far as classification performance is concerned.

³Later, we will describe some other methodologies.

The Need for Explanations In some applications, establishing the class of each example is not enough. Just as desirable is to know the reasons behind the classification. Thus a patient is unlikely to give consent to amputation if the only argument in support of surgery is, “this is what our computer says.” But how to find a better explanation?

In the “pies” domain, a lot can be gleaned from the boolean expression itself. For instance, we may notice that a pie was labeled as negative whenever its shape was square, and its filling white. Combining this observation with alternative sources of knowledge may offer useful insights: the dark shade of the filling may indicate poppy, an ingredient Johnny is known to love; or the crust of circular pies turns out to be more crispy than that of square ones; and so on. The knowledge obtained in this manner can be more desirable than the classification itself.

By contrast, the classifier in the “circles” domain is a mathematical expression that acts as a “black box” which accepts an example’s description and returns the class label without telling us anything else. This is not necessarily a shortcoming. In some applications, an explanation is nothing more than a welcome bonus; in others, it is superfluous. Consider a classifier that accepts a digital image of a hand-written character and returns the letter it represents. The user who expects several pages of text to be converted into a Word document will hardly insist on a detailed explanation for each single character.

Existence of Alternative Solutions By the way, we should notice that *many* apparently perfect classifiers can be induced from the given data. In the “pies” domain, the training set contained 12 examples, and the classes of the remaining 96 examples were unknown. Using some simple combinatorics, we realize that there are 2^{96} classifiers that label correctly all training examples but differ in the way they label the unknown 96. One induced classifier may label correctly every single future example—and another will misclassify them all.

What Have You Learned?

To make sure you understand the topic, try to answer the following questions. If needed, return to the appropriate place in the text.

- How can we estimate the error rate on examples that have not been seen during learning?
- Why is error rate usually higher on the testing set than on the training set?
- Give an example of a domain where the classifier also has to explain its action, and an example of a domain where this is unnecessary.
- What do we mean by saying that, “there is a combinatorial number of classifiers that correctly classify all training examples”?

1.5 Some Difficulties with Available Data

In some applications, the training set is created manually: an expert prepares the examples, tags them with class labels, chooses the attributes, and specifies the value of each attribute in each example. In other domains, the process is computerized. For instance, a company may want to be able to anticipate an employee's intention to leave. Their database contains, for each person, the address, gender, marital status, function, salary raises, promotions—as well as the information about whether the person is still with the company or, if not, the day they left. From this, a program can obtain the attribute vectors, labeled as positive if the given person left within a year since the last update of the database record.

Sometimes, the attribute vectors are automatically extracted from a database, and labeled by an expert. Alternatively, some examples can be obtained from a database, and others added manually. Often, two or more databases are combined. The number of such variations is virtually unlimited.

But whatever the source of the examples, they are likely to suffer from imperfections whose essence and consequences the engineer has to understand.

Irrelevant Attributes To begin with, some attributes are important, while others are not. While Johnny may be truly fond of poppy filling, his preference for a pie will hardly be driven by the cook's shoe size. This is something to be concerned about: *irrelevant* attributes add to computational costs; they can even mislead the learner. Can they be avoided?

Usually not. True, in manually created domains, the expert is supposed to know which attributes really matter, but even here, things are not so simple. Thus the author of the “pies” domain might have done her best to choose those attributes she believed to matter. But unsure about the real reasons behind Johnny's tastes, she may have included attributes whose necessity she suspected—but could not guarantee. Even more often the problems with relevance occur when the examples are extracted from a database. Databases are developed primarily with the intention to provide access to lots of information—of which usually only a tiny part pertains to the learning task. As to which part this is, we usually have no idea.

Missing Attributes Conversely, some critical attributes can be missing. Mindful of his parents' finances, Johnny may be prejudiced against expensive pies. The absence of attribute `price` will then make it impossible to induce a good classifier: two examples, identical in terms of the available attributes, can differ in the values of the vital “missing” attribute. No wonder that, though identically described, one example is positive, and the other is negative. When this happens, we say that the training set is *inconsistent*. The situation is sometimes difficult to avoid: not only may the expert be ignorant of the relevance of attribute `price`; it may be impossible to provide this attribute's values, and the attribute thus cannot be used anyway.

Redundant Attributes Somewhat less damaging are attributes that are *redundant* in the sense that their values can be obtained from other attributes. If the database contains a patient's `date-of-birth` as well as `age`, the latter is unnecessary

because it can be calculated by subtracting `date-of-birth` from today's date. Fortunately, redundant attributes are less dangerous than irrelevant or missing ones.

Missing Attribute Values In some applications, the user has no problems identifying the right choice of attributes. The problem is, however, that the value of some attributes are not known. For instance, the company analyzing the database of its employees may not know, for each person, the number of children.

Attribute: Value Noise Attribute values and class labels often cannot be trusted on account of unreliable sources of information, poor measurement devices, typos, the user's confusion, and many other reasons. We say that the data suffer from various kinds of *noise*.

Stochastic noise is random. For instance, since our body-weight varies during the day, the reading we get in the morning is different from the one in the evening. A human error can also play a part: lacking the time to take a patient's blood pressure, a negligent nurse simply scribbles down a modification of the previous reading. By contrast, *systematic noise* drags all values in the same direction. For instance, a poorly calibrated thermometer always gives a lower reading than it should. And something different occurs in the case of *arbitrary artifacts*; here, the given value bears no relation to reality such as when an EEG electrode gets loose and, from that moment on, all subsequent readings will be zero.

Class-Label Noise Class labels suffer from similar problems as attributes. The labels recommended by an expert may not have been properly recorded; alternatively, some examples find themselves in a "gray area" between two classes, in which event the correct labels are not certain. Both cases represent stochastic noise, of which the latter may affect negatively only examples from the borderline region between the two classes. However, class-label noise can also be systematic: a physician may be reluctant to diagnose a rare disease unless the evidence is overwhelming—his class labels are then more likely to be negative than positive. Finally, arbitrary artifacts in class labels are encountered in domains where the classes are supplied by an automated process that has gone wrong.

Class-label noise can be more dangerous than attribute-value noise. Thus in the "circles" domain, an example located deep inside the positive region will stay there even if an attribute's value is slightly modified; only the borderline example will suffer from being "sent across the border." By contrast, class-label noise will invalidate *any* example.

What Have You Learned?

To make sure you understand the topic, try to answer the following questions. If needed, return to the appropriate place in the text.

- Explain the following types of attributes: irrelevant, redundant, and missing. Illustrate each of them using the "pies" domain.

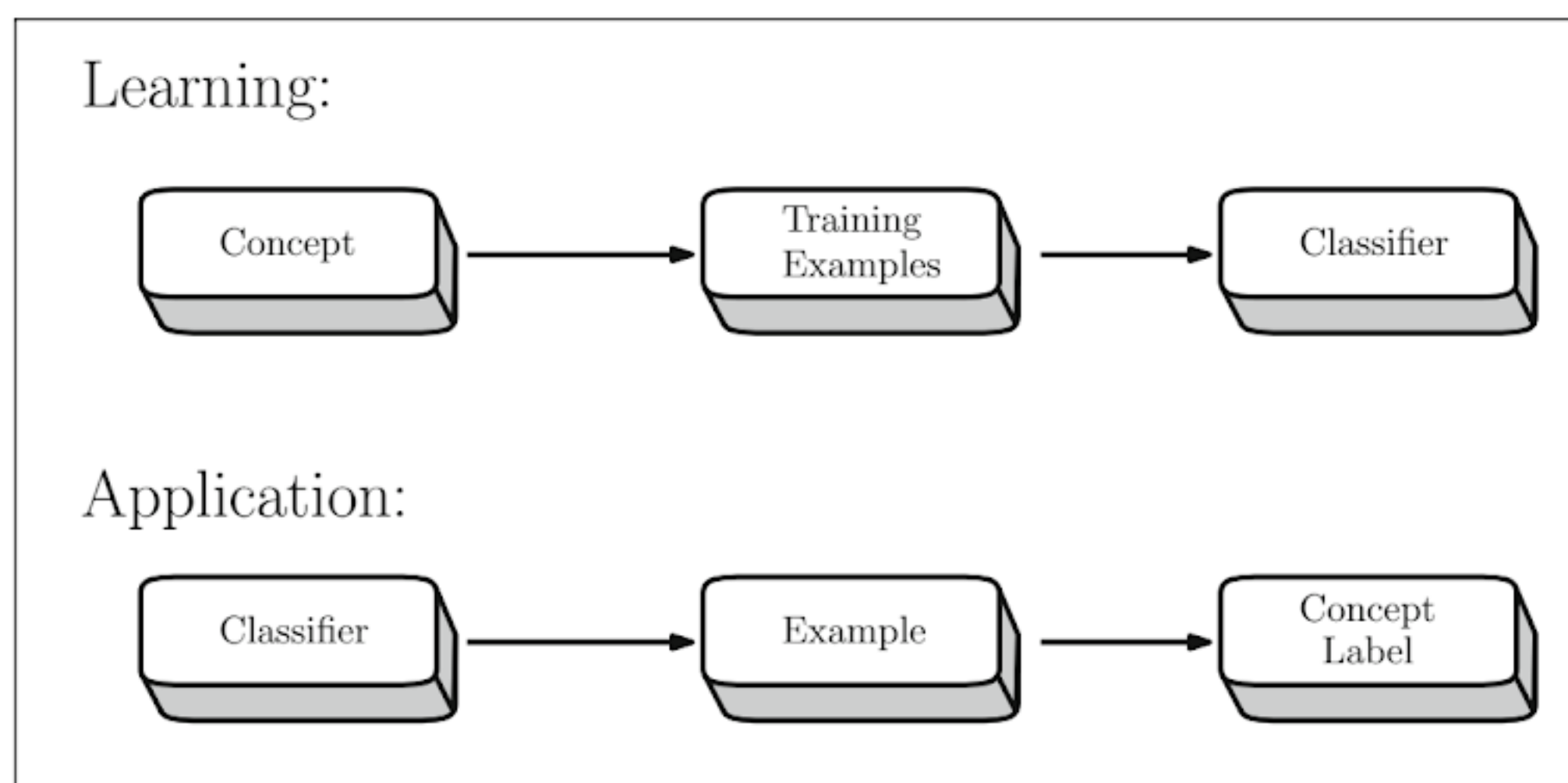


Fig. 1.7 The training examples are used to induce a classifier. The classifier is then employed to classify future examples

- What is meant by “inconsistent training set”? What can be the cause? How can it affect the learning process?
- What kinds of noise do we know? What are their possible sources?

1.6 Summary and Historical Remarks

- Induction from a training set of pre-classified examples is the most deeply studied machine-learning task.
- Historically, the task is cast as search. One can propose a mechanism that exploits the well-established search technique of hill climbing defined by an initial state, final state, interim states, search operators, and evaluation functions.
- Mechanical use of search is not the ultimate solution, though. The rest of the book will explore more useful techniques.
- Classifier performance is estimated with the help of pre-classified testing data. The simplest performance criterion is error rate, the percentage of examples misclassified by the classifier. The baseline scenario is shown in Fig. 1.7.
- Two classifiers that both correctly classify all training examples may differ significantly in their handling of the testing set.
- Apart from low error rate, some applications require that the classifier provides the reasons behind the classification.
- The quality of the induced classifier depends on training examples. The quality of the training examples depends not only on their choice, but also on the attributes used to describe them. Some attributes are relevant, others irrelevant or redundant. Quite often, critical attributes are missing.
- The attribute values and class labels may suffer from stochastic noise, systematic noise, and random artefacts. The value of an attribute in a concrete example may not be known.

Historical Remarks The idea of casting the machine-learning task as search was popular in the 1980s and 1990s. While several “founding fathers” came to see things this way independently of each other, Mitchell [67] is often credited with being the first to promote the search-based approach; just as influential, however, was the family of AQ-algorithms proposed by Michalski [59]. The discipline got a major boost by the collection of papers edited by Michalski et al. [61]. They framed the mindset of a whole generation.

There is much more to search algorithms. The interested reader is referred to textbooks of Artificial Intelligence, of which perhaps the most comprehensive is Russell and Norvig [84] or Coppin [17].

The reader may find it interesting that the question of proper representation of concepts or classes intrigued philosophers for centuries. Thus John Stuart Mill [65] explored concepts that are related to what the next chapter calls *probabilistic*

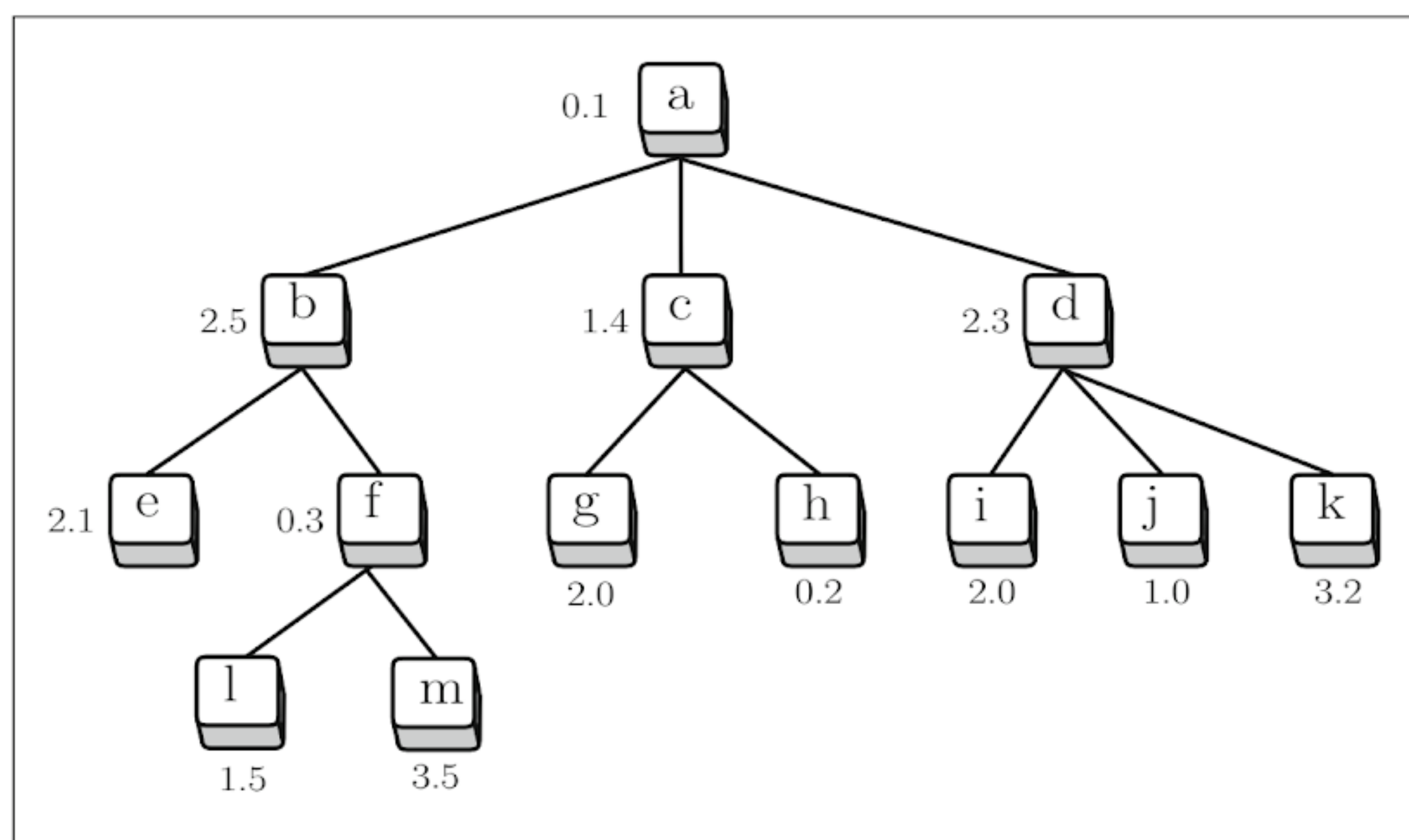


Fig. 1.8 Determine the order in which these search states are visited by heuristic search algorithms. The numbers next to the “boxes” give the values of the evaluation function for the individual search states

representation; and William Whewel [96] advocated *prototypical* representations that are close to the subject of our Chap. 3.

1.7 Solidify Your Knowledge

The exercises are to solidify the acquired knowledge. The suggested thought experiments will help the reader see this chapter’s ideas in a different light and provoke independent thinking. Computer assignments will force the readers to pay attention to seemingly insignificant details they might otherwise overlook.

Exercises

1. In the sliding-tiles puzzle, suggest a better evaluation function than the one used in the text.
2. Figure 1.8 shows a search tree where each node represents one search state and is tagged with the value of the evaluation function. In what order will these states be visited by hill-climbing search?
3. Suppose the evaluation function in the “pies” domain calculates the percentage of correctly classified training examples. Let the initial state be the expression describing the second positive example in Table 1.1. Hand-simulate the hill-climbing search that uses generalization and specialization operators.
4. What is the size of the instance space in a domain where examples are described by ten boolean attributes? How large is then the space of classifiers?

Give It Some Thought

1. In the “pies” domain, the size of the space of all classifiers is 2^{108} , provided that each subset of the instance space can be represented by a distinct classifier. How much will the search space shrink if we permit only classifiers in the form of conjunctions of attribute-value pairs?
2. What kind of noise can you think of in the “pies” domain? What can be the source of this noise? What other issues may render training sets of this kind less than perfect?
3. Some classifiers behave as black boxes that do not offer much in the way of explanations. This, for instance, was the case of the “circles” domain. Suggest examples of domains where black-box classifiers are impractical, and suggest domains where this limitation does not matter.
4. Consider the data-related difficulties summarized in Sect. 1.5. Which of them are really serious, and which can perhaps be tolerated?
5. What is the difference between redundant attributes and irrelevant attributes?
6. Take a class that you think is difficult to describe—for instance, the recognition of a complex biological object (oak tree, ostrich, etc.) or the recognition of a music genre (rock, folk, jazz, etc.). Suggest the list of attributes to describe the training examples. Are the values of these attributes easy to obtain? Which of the problems discussed in this chapter do you expect will complicate the learning process?

Computer Assignments

1. Write a program implementing hill climbing and apply it to the sliding-tiles puzzle. Choose appropriate representation for the search states, write a module that decides whether a state is a final state, and implement the search operators. Define two or three alternative evaluation functions and observe how each of them leads to a different sequence of search steps.
2. Write a program that will implement the “growing circles” algorithm from Sect. 1.3. Create a training set of two-dimensional examples such as those in Fig. 1.5. The learning program will use the hill-climbing search. The evaluation function will calculate the percentage of training examples correctly classified by the classifier. Consider the following search operators: (1) increase/decrease the radius of the circle, (2) use a different training example as the circle’s center.
3. Write a program that will implement the search for the description of the “pies that Johnny likes.” Define your own generalization and specialization operators. The evaluation function will rely on the error rate observed on the training examples.

Chapter 2

Probabilities: Bayesian Classifiers

The earliest attempts to predict an example’s class based on the known attribute values go back to well before World War II—prehistory, by the standards of computer science. Of course, nobody used the term “machine learning,” in those days, but the goal was essentially the same as the one addressed in this book.

Here is the essence of the solution strategy they used: using the Bayesian probabilistic theory, calculate for each class the probability of the given object belonging to it, and then choose the class with the highest value.

2.1 The Single-Attribute Case

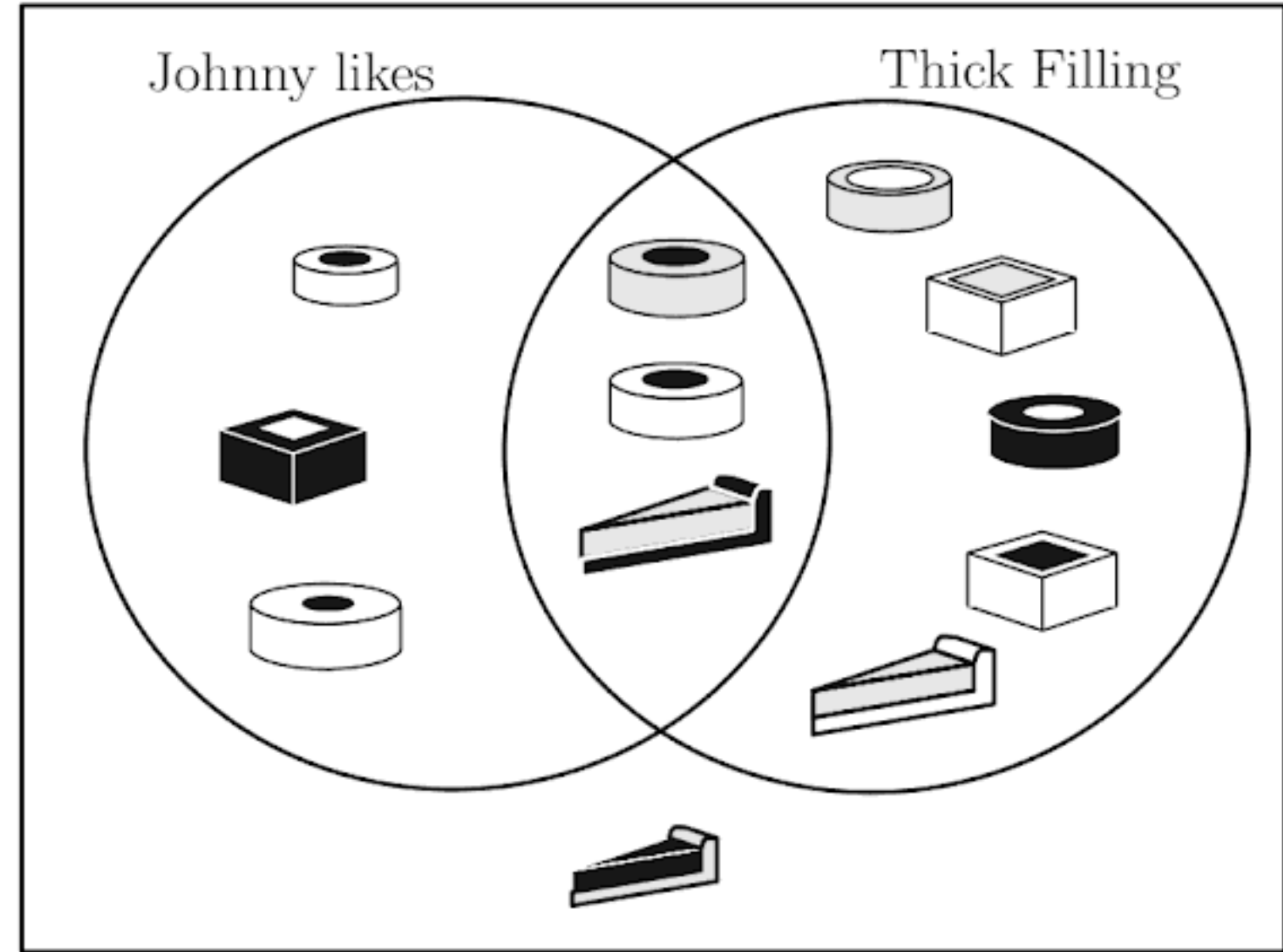
Let us start with something so simple as to be unrealistic: a domain where each example is described with a single attribute. Once we have developed the basic principles, we will generalize them so that they can be used in more practical domains.

Probabilities The basics are easily explained using the toy domain from the previous chapter. The training set consists of twelve pies ($N_{all} = 12$), of which six are positive examples of the given concept ($N_{pos} = 6$) and six are negative ($N_{neg} = 6$). Assuming that the examples represent faithfully the real situation, the probability of Johnny liking a randomly picked pie is therefore 50%:

$$P(\text{pos}) = \frac{N_{pos}}{N_{all}} = \frac{6}{12} = 0.5 \tag{2.1}$$

Let us now take into consideration one of the attributes, say, *filling-size*. The training set contains eight examples with thick filling ($N_{thick} = 8$). Out of these, three are labeled as positive ($N_{pos|thick} = 3$). This means that the “*conditional probability*

Fig. 2.1 The prior probabilities, $P(\text{pos}) = \frac{6}{12}$ and $P(\text{thick}) = \frac{8}{12}$; the conditional probabilities, $P(\text{pos}|\text{thick}) = \frac{3}{8}$ and $P(\text{thick}|\text{pos}) = \frac{3}{6}$; and the joint probability, $P(\text{likes}, \text{thick}) = \frac{3}{12}$



of an example being positive given that filling-size=thick” is 37.5%—this is what the relative frequency of positive examples among those with thick filling implies:

$$P(\text{pos}|\text{thick}) = \frac{N_{\text{pos}|\text{thick}}}{N_{\text{thick}}} = \frac{3}{8} = 0.375 \tag{2.2}$$

Applying Conditional Probability to Classification Importantly, the relative frequency is calculated only for pies with the given attribute value. Among these same eight pies, five represented the negative class, which means that $P(\text{neg}|\text{thick}) = 5/8 = 0.625$. Observing that $P(\text{neg}|\text{thick}) > P(\text{pos}|\text{thick})$, we conclude that the probability of Johnny disliking a pie with thick filling is greater than the probability of the opposite case. It thus makes sense for the classifier to label all examples with filling-size=thick as negative instances of the “pie that Johnny likes.”

Note that conditional probability, $P(\text{pos}|\text{thick})$, is more trustworthy than the prior probability, $P(\text{pos})$, because of the additional information that goes into its calculation. This is only natural. In a DayCare center where the number of boys is about the same as that of girls, we expect a randomly selected child to be a boy with $P(\text{boy}) = 0.5$. But the moment we hear someone call the child Johnny, we increase this expectation, knowing that it is rare for a girl to have this name. This is why $P(\text{boy}|\text{Johnny}) > P(\text{boy})$.

Joint Probability Conditional probability should not be confused with *joint probability* of two events occurring simultaneously. Be sure to use the right notation: in joint probability, the terms are separated by commas, $P(\text{pos}, \text{thick})$; in conditional probability, by a vertical bar, $P(\text{pos}|\text{thick})$. For a randomly picked pie, $P(\text{pos}, \text{thick})$ denotes the probability that the example is positive *and* its filling is thick; whereas $P(\text{pos}|\text{thick})$ refers to the occurrence of a positive example among those that have filling-size=thick.

A Concrete Example Figure 2.1 illustrates the terms. The rectangle represents all pies. The positive examples are contained in one circle and those with `filling-size=thick` in the other; the intersection contains three instances that satisfy both conditions; one pie satisfies neither, and is therefore left outside both circles. The conditional probability, $P(\text{pos}|\text{thick}) = 3/8$, is obtained by dividing the size of the intersection (three) by the size of the circle `thick` (eight). The joint probability, $P(\text{pos}, \text{thick}) = 3/12$, is obtained by dividing the size of the intersection (three) by the size of the entire training set (twelve). The prior probability of $P(\text{pos}) = 6/12$ is obtained by dividing the size of the circle `pos` (six) with that of the entire training set (twelve).

Obtaining Conditional Probability from Joint Probability The picture convinces us that joint probability can be obtained from prior probability and conditional probability:

$$P(\text{pos}, \text{thick}) = P(\text{pos}|\text{thick}) \cdot P(\text{thick}) = \frac{3}{8} \cdot \frac{8}{12} = \frac{3}{12}$$

$$P(\text{thick}, \text{pos}) = P(\text{thick}|\text{pos}) \cdot P(\text{pos}) = \frac{3}{6} \cdot \frac{6}{12} = \frac{3}{12}$$

Note that joint probability can never exceed the value of the corresponding conditional probability: $P(\text{pos}, \text{thick}) \leq P(\text{pos}|\text{thick})$. This is because conditional probability is multiplied by prior probability, $P(\text{thick})$ or $P(\text{pos})$, which can never be greater than 1.

Another fact to notice is that $P(\text{thick}, \text{pos}) = P(\text{pos}, \text{thick})$ because both represent the same thing: the probability of `thick` and `pos` co-occurring. Consequently, the left-hand sides of the previous two formulas have to be equal, which implies the following:

$$P(\text{pos}|\text{thick}) \cdot P(\text{thick}) = P(\text{thick}|\text{pos}) \cdot P(\text{pos})$$

Dividing both sides of this last equation by $P(\text{thick})$, we obtain the famous Bayes formula, the foundation for the rest of this chapter:

$$P(\text{pos}|\text{thick}) = \frac{P(\text{thick}|\text{pos}) \cdot P(\text{pos})}{P(\text{thick})} \quad (2.3)$$

If we derive the analogous formula for the probability that pies with `filling-size = thick` will belong to the negative class, we obtain the following:

$$P(\text{neg}|\text{thick}) = \frac{P(\text{thick}|\text{neg}) \cdot P(\text{neg})}{P(\text{thick})} \quad (2.4)$$

Comparison of the values calculated by these two formulas will tell us which class, `pos` or `neg`, is more probable. Things are simpler than they look: since the denominator, $P(\text{thick})$, is the same for both classes, we can just as well ignore it and simply choose the class for which the numerator is higher.

A Trivial Numeric Example That this formula leads to correct values is illustrated in Table 2.1 which, for the sake of simplicity, deals with the trivial case where the examples are described by a single boolean attribute. So simple is this single-attribute world, actually, that we might easily have obtained $P(\text{pos}|\text{thick})$ and $P(\text{neg}|\text{thick})$ directly from the training set, without having to resort to the mighty Bayes formula—this makes it easy to verify the correctness of the results.

When the examples are described by two or more attributes, the way of calculating the probabilities is essentially the same, but we need at least one more trick. This will be introduced in the next section.

What Have You Learned?

To make sure you understand the topic, try to answer the following questions. If needed, return to the appropriate place in the text.

- How is the Bayes formula derived from the relation between the conditional and joint probabilities?
- What makes the Bayes formula so useful? What does it enable us to calculate?
- Can the joint probability, $P(x, y)$, have a greater value than the conditional probability, $P(x|y)$? Under what circumstances is $P(x|y) = P(x, y)$?

2.2 Vectors of Discrete Attributes

Let us now proceed to the question how to apply the Bayes formula in domains where the examples are described by vectors of attributes such as $\mathbf{x} = (x_1, x_2, \dots, x_n)$.

Multiple Classes Many realistic applications have more than two classes, not just the `pos` and `neg` from the “pies” domain. If c_i is the label of the i -th class, and if \mathbf{x} is the vector describing the object we want to classify, the Bayes formula acquires the following form:

$$P(c_i|\mathbf{x}) = \frac{P(\mathbf{x}|c_i)P(c_i)}{P(\mathbf{x})}$$

Table 2.1 Illustrating the principle of Bayesian decision making

Let the training examples be described by a single attribute, `filling-size`, whose value is either `thick` or `thin`. We want the machine to recognize the positive class (`pos`). Here are the eight available training examples:

	ex1	ex2	ex3	ex4	ex5	ex6	ex7	ex8
Size	thick	thick	thin	thin	thin	thick	thick	thick
Class	pos	pos	pos	pos	neg	neg	neg	neg

The probabilities of the individual attribute values and class labels are obtained by their relative frequencies. For instance, three out of the eight examples are characterized by `filling-size=thin`; therefore, $P(\text{thin}) = 3/8$.

$$P(\text{thin}) = 3/8$$

$$P(\text{thick}) = 5/8$$

$$P(\text{pos}) = 4/8$$

$$P(\text{neg}) = 4/8$$

The conditional probability of a concrete attribute value within a given class is, again, determined by relative frequency. Our training set yields the following values:

$$P(\text{thin}|\text{pos}) = 2/4$$

$$P(\text{thick}|\text{pos}) = 2/4$$

$$P(\text{thin}|\text{neg}) = 1/4$$

$$P(\text{thick}|\text{neg}) = 3/4$$

Using these values, the Bayes formula gives the following conditional probabilities:

$$P(\text{pos}|\text{thin}) = 2/3$$

$$P(\text{pos}|\text{thick}) = 2/5$$

$$P(\text{neg}|\text{thin}) = 1/3$$

$$P(\text{neg}|\text{thick}) = 3/5$$

(note that $P(\text{pos}|\text{thin}) + P(\text{neg}|\text{thin}) = P(\text{pos}|\text{thick}) + P(\text{neg}|\text{thick}) = 1$)

Based on these results, we conclude that an example with `filling-size=thin` should be classified as positive because $P(\text{pos}|\text{thin}) > P(\text{neg}|\text{thin})$. Conversely, an example with `filling-size = thick` should be classified as negative because $P(\text{neg}|\text{thick}) > P(\text{pos}|\text{thick})$.

The denominator being the same for each class, we choose the class that maximizes the numerator, $P(\mathbf{x}|c_i)P(c_i)$. Here, $P(c_i)$ is easy to estimate by the relative frequency of c_i in the training set. As for $P(\mathbf{x}|c_i)$, however, things are not so simple.

- Under what circumstances shall we assume that the individual attributes are mutually independent? What benefit does this assumption bring for the estimates of $P(\mathbf{x}|c_i)$?
- Discuss the conflicting aspects of this assumption.

2.3 Probabilities of Rare Events: Exploiting the Expert’s Intuition

In the first approximation, probability is almost identified with relative frequency: having observed x thirty times in one hundred trials, we assume that $P(x) = 0.3$. This is how we did it in the previous sections.

Table 2.2 Bayesian classification: examples described by vectors of independent attributes

Suppose we want to apply the Bayesian formula to the training set from Table 1.1 in order to determine the class of the following object:

```
x = [shape=square, crust-size=thick, crust-shade=gray
      filling-size=thin, filling-shade=white]
```

There are two classes, pos and neg. The procedure is to calculate the numerator of the Bayes formula separately for each of them, and then choose the class with the higher value. In the training set, each class has the same number of representatives: $P(\text{pos}) = P(\text{neg}) = 0.5$. The remaining terms, $\prod_{i=1}^n P(x_i|\text{pos})$ and $\prod_{i=1}^n P(x_i|\text{neg})$, are calculated from the following conditional probabilities:

$P(\text{shape=square} \text{pos})$	$= 1/6$	$P(\text{shape=square} \text{neg})$	$= 2/6$
$P(\text{crust-size=thick} \text{pos})$	$= 5/6$	$P(\text{crust-size=thick} \text{neg})$	$= 5/6$
$P(\text{crust-shade=gray} \text{pos})$	$= 1/6$	$P(\text{crust-shade=gray} \text{neg})$	$= 2/6$
$P(\text{filling-size=thin} \text{pos})$	$= 3/6$	$P(\text{filling-size=thin} \text{neg})$	$= 1/6$
$P(\text{filling-shade=white} \text{pos})$	$= 1/6$	$P(\text{filling-shade=white} \text{neg})$	$= 2/6$

Based on these values, we obtain the following probabilities:

$$P(\mathbf{x}|\text{pos}) = \prod_{i=1}^n P(x_i|\text{pos}) = \frac{1}{6} \cdot \frac{5}{6} \cdot \frac{1}{6} \cdot \frac{3}{6} \cdot \frac{1}{6} = \frac{15}{6^5}$$

$$P(\mathbf{x}|\text{neg}) = \prod_{i=1}^n P(x_i|\text{neg}) = \frac{2}{6} \cdot \frac{5}{6} \cdot \frac{2}{6} \cdot \frac{1}{6} \cdot \frac{2}{6} = \frac{40}{6^5}$$

Since $P(\mathbf{x}|\text{pos}) < P(\mathbf{x}|\text{neg})$, we label \mathbf{x} with the negative class.

Table 2.3 Classification with the Naive-Bayes principle

The example to be classified is described by $\mathbf{x} = (x_1, \dots, x_n)$.

1. For each x_i , and for each class c_j , calculate the conditional probability, $P(x_i|c_j)$, as the relative frequency of x_i among those training examples that belong to c_j .
2. For each class, c_j , carry out the following two steps:
 - i) estimate $P(c_j)$ as the relative frequency of this class in the training set;
 - ii) calculate the conditional probability, $P(\mathbf{x}|c_j)$, using the “naive” assumption of mutually independent attributes:

$$P(\mathbf{x}|c_j) = \prod_{i=1}^n P(x_i|c_j)$$

3. Choose the class with the highest value of $P(c_j) \cdot \prod_{i=1}^n P(x_i|c_j)$.
-

To be fair, though, such estimates can be trusted only when supported by a great many observations. It is conceivable that a coin flipped four times comes up heads three times, and yet it will be overhasty to interpret this observation as meaning that $P(\text{heads}) = 0.75$; the physics of the experiment suggests that a fair coin should come up heads 50% of the time. Can this *prior expectation* help us improve probability estimates in domains with insufficient numbers of observations?

The answer is, “Yes, we can use the *m*-estimate.”

The Essence of an *m*-Estimate Let us illustrate the principle using the case of an unfair coin where one side comes up somewhat more frequently than the other. In the absence of any better guidance, the prior expectation of heads is $\pi_{\text{heads}} = 0.5$. An auxiliary parameter, *m*, helps the engineer tell the class-predicting program *how confident* he is in this value, how much the prior expectation can be trusted (higher *m* indicating higher confidence).

Let us denote by N_{all} the number of times the coin was flipped, and by N_{heads} the number of times the coin came up heads. The way to combine these values with the prior expectation and confidence is summarized by the following formula:

$$P_{\text{heads}} = \frac{N_{\text{heads}} + m\pi_{\text{heads}}}{N_{\text{all}} + m} \quad (2.7)$$

Note that the formula degenerates to the prior expectation, π_{heads} , if $N_{\text{all}} = N_{\text{heads}} = 0$. Conversely, it converges to that of relative frequency if N_{all} and N_{heads} are so large as to render the terms $m\pi_{\text{heads}}$ and m negligible. Using the values $\pi_{\text{heads}} = 0.5$ and $m = 2$, we obtain the following:

$$P_{\text{heads}} = \frac{N_{\text{heads}} + 2 \times 0.5}{N_{\text{all}} + 2} = \frac{N_{\text{heads}} + 1}{N_{\text{all}} + 2}$$

Illustrating Probability Estimates Table 2.4 shows how the values thus calculated gradually evolve in the course of five trials. The reader can see that the *m*-estimate is for small numbers of experiments more in line with common sense than relative

Table 2.4 For each successive trial, the second row gives the observed outcome; the third, the relative frequency of *heads*; the last, the m -estimate of the probability, assuming $\pi_{heads} = 0.5$ and $m = 2$

Toss number	1	2	3	4	5
Outcome	Heads	Heads	Tails	Heads	Tails
Relative frequency	1.00	1.00	0.67	0.75	0.60
m -estimate	0.67	0.75	0.60	0.67	0.57

frequency. Thus after two trials, m -estimate suggests a 0.75 chance of heads, whereas anybody espousing relative frequency will have to concede that, based on the two experiments, there is a zero chance that the coin will come up tails. As the number of trials increases, though, the values returned by m -estimate and relative frequency tend to converge.

The Impact of the User's Confidence Let us take a closer look at the effect of m , the user's confidence. A lot is revealed if we compare the two different settings below: $m = 100$ on the left and $m = 1$ on the right (in both cases, $\pi_{heads} = 0.5$).

$$\frac{N_{heads} + 50}{N_{all} + 100} \qquad \frac{N_{heads} + 0.5}{N_{all} + 1}$$

The version with $m = 100$ allows the prior estimate to be modified only if really substantial evidence is available ($N_{heads} \gg 50, N_{all} \gg 100$). By contrast, the version with $m = 1$ allows the user's opinion to be controverted with just a few experimental trials.

Domains with More Than Two Outcomes Although we have used a two-outcome domain, the formula is applicable also in multi-outcome domains. Rolling a fair die can result in six different outcomes, and we expect that the probability of seeing, say, three points is $\pi_{three} = 1/6$. Using $m = 6$, we obtain the following:

$$P_{three} = \frac{N_{three} + m\pi_{three}}{N_{all} + m} = \frac{N_{three} + 6 \cdot \frac{1}{6}}{N_{all} + 6} = \frac{N_{three} + 1}{N_{all} + 6}$$

Again, if N_{all} is so high that $m = 6$ and $m\pi_{three} = 1$ can be neglected, the formula converges to relative frequency: $P_{three} = \frac{N_{three}}{N_{all}}$. If we do not want this to happen prematurely (perhaps because we have high confidence in the prior estimate, π_{three}), we prevent it by choosing a higher m .

The Limits of m -Estimates We should not forget that the m -estimate is only as good as the parameters it relies on. If we start from an unrealistic prior estimate, the result can be disappointing. Suppose that $\pi_{heads} = 0.9$ and $m = 10$. Equation (2.7) then turns into the following:

$$P_{heads} = \frac{N_{heads} + 9}{N_{all} + 10}$$

When we use this formula to recalculate the values from Table 2.4, we will realize that, after five trials, the probability is estimated as $P_{heads} = \frac{3+9}{5+10} = \frac{12}{15} = 0.8$, surely a less plausible value than the one obtained in the case of $\pi_{heads} = 0.5$ where we got $P_{heads} = 0.57$. The reader is encouraged to verify that the situation will somewhat improve if we reduce m .

Mathematical Soundness Let us make one last comment. A common understanding in mathematics is that the probabilities of all possible events should sum up to 1: if an experiment can have N different outcomes, and if P_i is the probability of the i -th outcome, then $\sum_{i=1}^N P_i = 1$. It is easy to verify that Eq. (2.7) satisfies this condition for any value of m . Suppose we are dealing with the coin-tossing domain where there are only two possible outcomes. If the prior estimates sum up to 1 ($\pi_{heads} + \pi_{tails} = 1$), then, given that $N_{heads} + N_{tails} = N_{all}$, we derive the following:

$$\begin{aligned} P_{heads} + P_{tails} &= \frac{N_{heads} + m\pi_{heads}}{N_{all} + m} + \frac{N_{tails} + m\pi_{tails}}{N_{all} + m} \\ &= \frac{N_{heads} + N_{tails} + m(\pi_{heads} + \pi_{tails})}{N_{all} + m} = 1 \end{aligned}$$

The interested reader will easily generalize this to any finite number of classes.

Why This May Be Useful In the problem presented in Table 2.5, we want to classify example \mathbf{x} using the Bayesian classifier. To be able to do that, we first need to calculate the requisite conditional probabilities. Trying to do so for the positive class, however, we realize that, since the training set is so small, none of the training examples has `crust-shade=gray`, the value observed in \mathbf{x} . If the probabilities are estimated by relative frequency, this concrete conditional probability would be 0. As a result, $P(\mathbf{x}|\text{pos}) = 0$, regardless of all the other probabilities. This simply does not seem right.

The problem disappears if we use m -estimate instead of relative frequency because the m -estimate is non-zero even if the concrete value has never being observed in the training set.

What Have You Learned?

To make sure you understand the topic, try to answer the following questions. If needed, return to the appropriate place in the text.

- Under what circumstances is relative frequency ill-suited for estimates of discrete probabilities?
- What is the impact of parameter m in Eq. (2.7)? Under what circumstances will you prefer large m , and when will you rather go for small m ?
- What is the impact of the prior estimate, π_{heads} , in Eq. (2.7)? How is the credibility of m -estimates affected by unrealistic values of π_{heads} ?

Table 2.5 An example of one reason for using m -estimates in Bayesian classification

Let us return to the “pies” domain from Table 1.1. Remove from the table the first example, then use the rest for the calculation of the probabilities.

```
x = [shape=circle, crust-size=thick, crust-shade=gray
      filling-size=thick, filling-shade=dark]
```

Let us first calculate the probabilities of the individual attribute values:

$P(\text{shape}=\text{circle} \text{pos})$	$= 3/5$
$P(\text{crust-size}=\text{thick} \text{pos})$	$= 4/5$
$P(\text{crust-shade}=\text{gray} \text{pos})$	$= 0/5$
$P(\text{filling-size}=\text{thick} \text{pos})$	$= 2/5$
$P(\text{filling-shade}=\text{dark} \text{pos})$	$= 3/5$

Based on these values, we obtain the following probabilities:

$$P(x|\text{pos}) = \frac{3 \times 4 \times 0 \times 2 \times 3}{5^5} = 0.$$

We see that the circumstance that none of the five positive examples has `crust-shade=gray` causes the corresponding conditional probability to equal 0.

The problem is solved if we calculate the probabilities using the m -estimate. In this case, none of the conditional probabilities will be 0.

2.4 How to Handle Continuous Attributes

Up till now, we limited our considerations to attributes that assume *discrete* values, estimating their probabilities either by relative frequency or by the m -estimate. This, however, is not enough. In many applications, we encounter attributes (such as age, price or weight) that acquire values from continuous domains.

Relative frequency is then impractical. While it is easy to establish that the probability of an engineering student being male is $P_{\text{male}} = 0.7$, the probability that this student’s body weight is 184.5 pounds cannot be specified so readily: the number of different weight values being infinite, the probability of any one of them is infinitesimally small. What to do in this case?

Discretizing Continuous Attributes One possibility is to *discretize*. The simplest “trick” will split the attribute’s original domain in two; for instance, by replacing age with the boolean attribute `old` that is *true* for $\text{age} > 60$ and *false* otherwise. However, at least part of the available information then gets lost: a person may be `old`, but we no longer know *how old*; nor do we know whether one `old` person is older than another `old` person.

What Have You Learned?

To make sure you understand the topic, try to answer the following questions. If needed, return to the appropriate place in the text.

- What is the probability density function, *pdf*, and how does it help us in the context of Bayesian classification?
- Explain the discretization mechanism that helped us arrive at an informal definition of a *pdf*.
- How does the Bayes formula change in domains with continuous attributes? How do we estimate the values of the individual terms?

2.5 Gaussian “Bell” Function: A Standard *pdf*

One way to approximate a *pdf* is to employ the discretization technique from the previous section. Alternatively, we can capitalize on standardized models known to be applicable to many realistic situations. Perhaps the most popular among them is the *gaussian function*, named after the great German mathematician.

The Shape and the Formula Describing It The curve in Fig. 2.3 is an example; its shape betrays why many people call it a “bell function.” The maximum is reached at the mean, $x = \mu$, and the curve slopes down gracefully with the growing distance of x from μ . It is reasonable to expect that this is a good model of the *pdf* of such variables as the body temperature where the density peaks around $x = 99.7$ degrees Fahrenheit.

Expressed mathematically, the gaussian function is defined by the following formula where e is the base of the natural logarithm:

$$p(x) = k \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (2.10)$$

Parameters Note that the greater the difference between x and μ , the greater the exponent’s numerator, and thus the smaller the value of $p(x)$ because the exponent is negative. The reason the numerator is squared, $(x - \mu)^2$, is to make sure that the value slopes down with the same angle on both sides of the mean, μ ; the curve is symmetric. How steep the slope is depends on σ^2 , a parameter called *variance*. Greater variance means smaller sensitivity to the difference between x and μ , and thus a “flatter” bell curve; conversely, smaller variance defines a narrower bell curve.

The task for the coefficient k is to make the area under the bell function equal to 1 as required by the theory of probability. It would be relatively easy to prove that this happens when k is determined by the following formula:

$$k = \frac{1}{\sqrt{2\pi\sigma^2}} \quad (2.11)$$

Setting the Parameter Values To be able to use this model when approximating $p_{c_i}(x)$ in a concrete application, we only need to estimate the values of its parameters, μ and σ^2 . This is easy. Suppose that class c_i has m representatives among the training examples. If x_i is the value of the given attribute in the i -th example, then the mean and variance, respectively, are calculated using the following formulas:

$$\mu = \frac{1}{m} \sum_{i=1}^m x_i \quad (2.12)$$

$$\sigma^2 = \frac{1}{m-1} \sum_{i=1}^m (x_i - \mu)^2 \quad (2.13)$$

In plain English, the gaussian center, μ , is obtained as the arithmetic average of the values observed in the training examples, and the variance is obtained as the average of the squared differences between x_i and μ . Note that, when calculating variance, we divide the sum by $m-1$, and not by m , as we might expect. The intention is to compensate for the fact that μ itself is only an estimate. The variance should therefore be somewhat higher than what it would be if we divided by m . Of course, this matters only if the training set is small: for large m , the difference between m and $m-1$ is negligible.

What Have You Learned?

To make sure you understand the topic, try to answer the following questions. If needed, return to the appropriate place in the text.

- Give an example of a continuous variable whose *pdf* can be expected to follow the gaussian distribution.
- What parameters define the bell function? How can we establish their values using the training set?
- How—and why—do we normalize the bell function?

2.6 Approximating PDFs with Sets of Gaussians

While the bell function represents a good mechanism to approximate the *pdf* in many realistic domains, it is not a panacea. Some variables simply do not behave that way. Just consider the distribution of `body-weight` in a group that mixes grade-school children with their parents. If we create the *pdf* using the discretization method, we will observe two peaks: one for the kids, and the other for the grown-

ups. There may be three peaks if it turns out that `body-weight` of fathers is distributed around a higher mean than that of the mothers. And the number of peaks can be higher still if the families come from diverse ethnic groups.

Combining Gaussian Functions No doubt, a single bell function would misrepresent the situation. But what if we combine two or more of them? If we knew the diverse sources of the examples, we might create a separate gaussian for each source, and then superimpose the bell functions on each other. Would this solve our problem?

The honest answer is, “yes, in this ideal case.” In reality, though, prior knowledge about diverse sources is rarely available. A better solution will divide the `body-weight` values into great many random groups. In the extreme, we may even go as far as to make each example a “group” of its own, and then identify a gaussian center with this example’s `body-weight`, thus obtaining m bell functions (for m examples).

The Formula to Combine Them Suppose we want to approximate the *pdf* of a continuous attribute, x . If we denote by μ_i the value of x in the i -th example, then the *pdf* is approximated by the following sum of m functions:

$$p(x) = k \cdot \sum_{i=1}^m e^{-\frac{(x-\mu_i)^2}{2\sigma^2}} \quad (2.14)$$

As before, the normalization constant, k , is here to make sure that the area under the curve is 1. This is achieved when k is calculated as follows:

$$k = \frac{1}{m\sigma\sqrt{2\pi}} \quad (2.15)$$

From mathematics, we know that if m is sufficiently high, Eq. (2.14) approximates the *pdf* with almost arbitrary accuracy.

Illustrating the Point Figure 2.4 illustrates the approach using a training set consisting of $m = 3$ examples, the values of attribute x being $x_1 = 0.4$, $x_2 = 0.5$ and $x_3 = 0.7$. The upper three charts show three bell functions, each centered at one of these points, the variance always being $\sigma^2 = 1$. The bottom chart shows the composed *pdf* created by putting together Eqs. (2.14) and (2.15), using the means, $\mu_1 = 0.4$, $\mu_2 = 0.5$, and $\mu_3 = 0.7$, and $\sigma^2 = 1$:

$$p(x) = \frac{1}{3\sqrt{2\pi}} \cdot \left[e^{-\frac{(x-0.4)^2}{2}} + e^{-\frac{(x-0.5)^2}{2}} + e^{-\frac{(x-0.7)^2}{2}} \right]$$

The Impact of Concrete Parameter Values The practical utility of the *pdf* thus obtained (its success when used in the Bayes formula) depends on the choice of σ^2 . In Fig. 2.4, we used $\sigma^2 = 1$, but there is no guarantee that this will work in any future application. To be able to adjust it properly, we need to understand how it affects the shape of the composite *pdf*.

Inspecting the gaussian formula, we realize that the choice of a very small value of σ^2 causes great sensitivity to the difference between x and μ_i ; the individual bell functions will be “narrow,” and the resulting *pdf* will be marked by steep peaks separated by extended “valleys.” Conversely, the consequence of a high σ^2 will be an almost flat *pdf*. Seeking a compromise between the two extremes, we will do well if we make σ^2 dependent on the distances between examples.

The simplest solution will use $\sigma^2 = \mu_{max} - \mu_{min}$, where μ_{max} and μ_{min} are the maximum and minimum values of μ_i , respectively. If you think this too crude, you may consider normalizing the difference by the number of examples: $\sigma^2 = (\mu_{max} - \mu_{min})/m$. Large training sets (with high m) will then lead to smaller variations that will narrow the contributing gaussians. Finally, in some domains we might argue that each of the contributing bell functions should have a variance of its own, proportional to the distance from the center of the nearest other bell function. In this case, however, we are no longer allowed to set the value of k by Eq. (2.15).

A Numeric Example The example in Table 2.6 illustrates the whole procedure on a concrete case of a small training set and a vector to be classified. The reader is encouraged to go through all its details to get used to the way the formulas are put together. See also the illustration in Fig. 2.5.

When There Are Too Many Examples For a training set of realistic size, it is impractical to identify each training example with one gaussian centers; nor is it necessary. More often than not, the examples are grouped in *clusters* that can be detected by *cluster analysis* techniques—see Chap. 14. Once the clusters have been found, we identify the gaussian centers with the centroids of the clusters.

What Have You Learned?

To make sure you understand the topic, try to answer the following questions. If needed, return to the appropriate place in the text.

- Under what circumstances is the gaussian function a poor model of the *pdf*?
- Why does the composite *pdf* have to be normalized by k ?
- How do we establish the centers and variances of the individual bell functions?

2.7 Summary and Historical Remarks

- Bayesian classifiers calculate the product $P(\mathbf{x}|c_i)P(c_i)$ separately for each class, c_i , and then label the example, \mathbf{x} , with the class where this product has the highest value.
- The main problem is how to calculate the probability, $P(\mathbf{x}|c_i)$. Most of the time, the job is simplified by making the assumption that the individual attributes are mutually independent, in which case $P(\mathbf{x}|c_i) = \prod_{j=1}^n P(x_j|c_i)$, where n is the number of attributes.

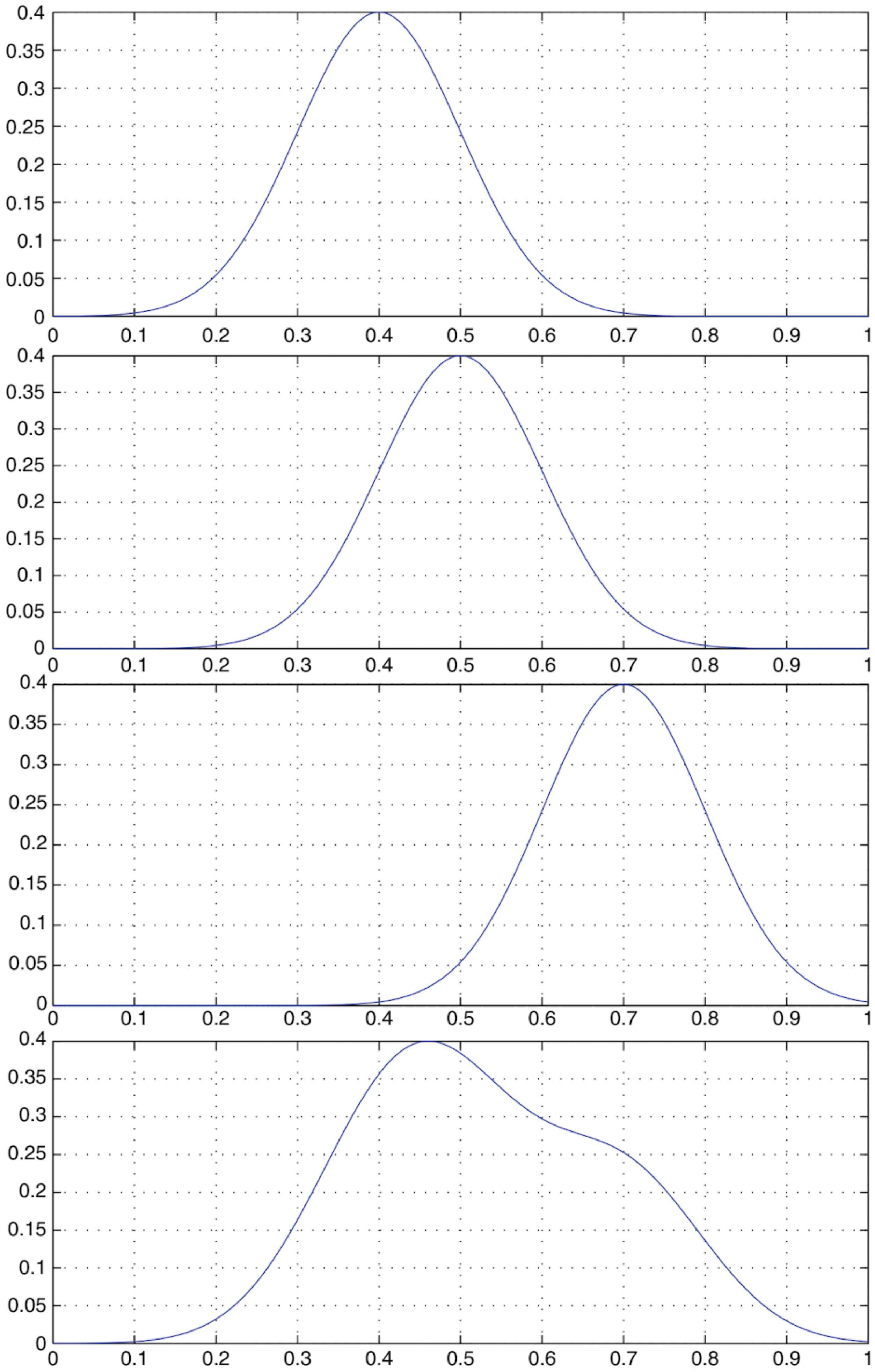


Fig. 2.4 Composing the *pdf* from three examples with the following values of attribute x : $\mu_1 = 0.4$, $\mu_2 = 0.5$, and $\mu_3 = 0.7$. The *upper three charts* show the contributing gaussians; the *bottom chart*, the composition. The variance is $\sigma^2 = 1$

[14]. The first to use the assumption of independent attributes was Good [33]. The idea of approximating *pdf*'s by the sum of bell functions comes from Parzen [74].

When provided with perfect information about the probabilities, the Bayesian classifier is guaranteed to provide the best possible classification accuracy. This is why it is sometimes used as a reference to which the performance of other approaches is compared.

2.8 Solidify Your Knowledge

The exercises are to solidify the acquired knowledge. The suggested thought experiments will help the reader see this chapter's ideas in a different light and provoke independent thinking. Computer assignments will force the readers to pay attention to seemingly insignificant details they might otherwise overlook.

Exercises

1. A coin tossed three times came up *heads*, *tails*, and *tails*, respectively. Calculate the *m*-estimate for these outcomes, using $m = 3$ and $\pi_{heads} = \pi_{tails} = 0.5$.
2. Suppose you have the following training examples, described by three attributes, x_1, x_2, x_3 , and labeled by classes c_1 and c_2 .

x_1	x_2	x_3	Class
2.1	0.2	3.0	c_1
3.3	1.0	2.9	c_1
2.7	1.2	3.4	c_1
0.5	5.3	0.0	c_2
1.5	4.7	0.5	c_2

Using these data, do the following:

- (a) Assuming that the attributes are mutually independent, approximate the following probability density functions: $p_{c_1}(\mathbf{x}), p_{c_2}(\mathbf{x}), p(\mathbf{x})$. Hint: use the idea of superimposed bell functions.
- (b) Using the *pdf*'s from the previous step, decide whether $\mathbf{x} = [1.4, 3.3, 3.0]$ should belong to c_1 or c_2 .

Give It Some Thought

1. How would you apply the m -estimate in a domain with three possible outcomes, $[A, B, C]$, each with the same prior probability estimate, $\pi_A = \pi_B = \pi_C = 1/3$? What if you trust your expectations of A but are not so sure about B and C ? Is there a way to reflect this circumstance in the value of the parameter m ?
2. Suggest the circumstances under which the accuracy of probability estimates will benefit from the assumption that attributes are mutually independent. Explain the advantages and disadvantages.
3. How would you calculate the probabilities of the output classes in a domain where some attributes are boolean, others discrete, and yet others continuous? Discuss the possibilities of combining different approaches.

Computer Assignments

1. Machine learning researchers often test their algorithms using publicly available benchmark domains. A large repository of such domains can be found at the following address: www.ics.uci.edu/~mlearn/MLRepository.html. Take a look at these data and see how they differ in the numbers of attributes, types of attributes, sizes and so on.
2. Write a computer program that will use the Bayes formula to calculate the class probabilities in a domain where all attributes are discrete. Apply this program to our “pies” domain.
3. For the case of continuous attributes, write a computer program that accepts the training examples in the form of a table such as the one in Exercise 3 above. Based on these, the program approximates the $pdfs$, and then uses them to determine the class labels of future examples.
4. Apply this program to a few benchmark domains from the UCI repository (choose from those where all attributes are continuous) and observe that the program succeeds in some domains better than in others.